



BBC MICRO COMPUTER SYSTEM

MICROCOMPUTER SYSTEM

# CP/M® 2.2 with GSX™ Graphics

for the BBC Microcomputer with Z80 second processor



BSP12

19. 2. 2012  
in  
the  
water



**CP/M®**  
Operating System  
**Manual**

PETERBOROUGH REGIONAL COLLEGE

PARK CRESCENT

PETERBOROUGH PE1 4DZ

9913

## COPYRIGHT

Copyright © 1976, 1977, 1978, 1979, 1982, and 1983 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

This manual is, however, tutorial in nature. Thus, the reader is granted permission to include the example programs, either in whole or in part, in his own programs.

## DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

## TRADEMARKS

CP/M and CP/NET are registered trademarks of Digital Research. ASM, DESPOOL, DDT, LINK-80, MAC, MP/M, PL/I-80 and SID are trademarks of Digital Research. Intel is a registered trademark of Intel Corporation. TI Silent 700 is a trademark of Texas Instruments Incorporated. Zilog and Z80 are registered trademarks of Zilog, Inc.

The *CP/M Operating System Manual* was printed in the United States of America.

First Edition: 1976

Second Edition: July 1982

Third Edition: September 1983

Neither the whole nor any part of the information contained in, or the product described in, this manual may be adapted or reproduced in any material form except with the prior written approval of Acorn Computers Limited (Acorn Computers).

The product described in this manual and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:

Technical Enquiries  
Acorn Computers Limited  
Fulbourn Road  
Cherry Hinton  
Cambridge CB1 4JN

Published by Acorn Computers Limited  
Part number 409001  
Issue number 1  
Date February 1984

# Table of Contents

## 1 CP/M Features and Facilities

1.1	Introduction .....	1-1
1.2	Functional Description .....	1-3
1.2.1	General Command Structure .....	1-3
1.2.2	File References .....	1-4
1.3	Switching Disks .....	1-7
1.4	Built-in Commands .....	1-7
1.4.1	ERA .....	1-8
1.4.2	DIR .....	1-9
1.4.3	REN .....	1-10
1.4.4	SAVE .....	1-11
1.4.5	TYPE .....	1-11
1.4.6	USER .....	1-12
1.5	Line Editing and Output Control .....	1-12
1.6	Transient Commands .....	1-14
1.6.1	STAT .....	1-15
1.6.2	ASM .....	1-22
1.6.3	LOAD .....	1-24
1.6.4	PIP .....	1-25
1.6.5	ED .....	1-35
1.6.6	SYSGEN .....	1-37
1.6.7	SUBMIT .....	1-39
1.6.8	DUMP .....	1-41
1.6.9	MOVCPM .....	1-42
1.7	BDOS Error Messages .....	1-44
1.8	Operation of CP/M on the MDS .....	1-46

## 2 ED

2.1	Introduction to ED .....	2-1
2.1.1	ED Operation .....	2-1
2.1.2	Text Transfer Functions .....	2-3
2.1.3	Memory Buffer Organization .....	2-4
2.1.4	Line Numbers and ED Start Up .....	2-5
2.1.5	Memory Buffer Operation .....	2-7
2.1.6	Command Strings .....	2-8

# Table of Contents (continued)

2.1.7	Text Search and Alteration .....	2-11
2.1.8	Source Libraries .....	2-15
2.1.9	Repetitive Command Execution.....	2-17
2.2	ED Error Conditions.....	2-18
2.3	Control Characters and Commands .....	2-19
<b>3</b>	<b>CP/M Assembler</b>	
3.1	Introduction.....	3-1
3.2	Program Format .....	3-3
3.3	Forming the Operand .....	3-4
3.3.1	Labels .....	3-5
3.3.2	Numeric Constants .....	3-5
3.3.3	Reserved Words .....	3-6
3.3.4	String Constants.....	3-7
3.3.5	Arithmetic and Logical Operators .....	3-7
3.3.6	Precedence of Operators .....	3-9
3.4	Assembler Directives.....	3-10
3.4.1	The ORG Directive .....	3-11
3.4.2	The END Directive.....	3-11
3.4.3	The EQU Directive.....	3-12
3.4.4	The SET Directive .....	3-13
3.4.5	The IF and ENDIF Directive .....	3-13
3.4.6	The DB Directive .....	3-15
3.4.7	The DW Directive.....	3-15
3.4.8	The DS Directive .....	3-16
3.5	Operation Codes .....	3-16
3.5.1	Jumps, Calls, and Returns .....	3-17
3.5.2	Immediate Operand Instructions .....	3-19
3.5.3	Increment and Decrement Instructions .....	3-20
3.5.4	Data Movement Instructions .....	3-21
3.5.5	Arithmetic Logic Unit Operations .....	3-22
3.5.6	Control Instructions .....	3-24
3.6	Error Messages.....	3-24
3.7	A Sample Session .....	3-26

# Table of Contents (continued)

## 4 CP/M Dynamic Debugging Tool

4.1	Introduction.....	4-1
4.2	DDT Commands.....	4-4
4.2.1	The A (Assembly) Command .....	4-4
4.2.2	The D (Display) Command.....	4-5
4.2.3	The F (Fill) Command .....	4-5
4.2.4	The G (Go) Command.....	4-6
4.2.5	The I (Input) Command.....	4-7
4.2.6	The L (List) Command .....	4-7
4.2.7	The M (Move) Command.....	4-8
4.2.8	The R (Read) Command .....	4-8
4.2.9	The S (Set) Command .....	4-9
4.2.10	The T (Trace) Command.....	4-9
4.2.11	The U (Untrace) Command.....	4-10
4.2.12	The X (Examine) Command.....	4-10
4.3	Implementation Notes .....	4-11
4.4	A Sample Program .....	4-12

## 5 CP/M 2 System Interface

5.1	Introduction.....	5-1
5.2	Operating System Call Conventions .....	5-4
5.3	A Sample File-to-File Copy Program.....	5-36
5.4	A Sample File Dump Utility .....	5-40
5.5	A Sample Random Access Program.....	5-46
5.6	System Function Summary.....	5-54

## 6 CP/M Alteration

6.1	Introduction.....	6-1
6.2	First-level System Regeneration .....	6-3
6.3	Second-level System Generation .....	6-6
6.4	Sample GETSYS and PUTSYS Program .....	6-11
6.5	Disk Organization .....	6-13
6.6	The BIOS Entry Points .....	6-15
6.7	A Sample BIOS .....	6-25
6.8	A Sample Cold Start Loader.....	6-25

# Table of Contents (continued)

6.9	Reserved Locations in Page Zero.....	6-26
6.10	Disk Parameter Tables .....	6-28
6.11	The DISKDEF Macro Library .....	6-34
6.12	Sector Blocking and Deblocking .....	6-39

## Appendices

A	The MDS Basic I/O System (BIOS) .....	A-1
B	A Skeletal CBIOS .....	B-1
C	A Skeletal GETSYS/PUTSYS Program .....	C-1
D	The MDS-800 Cold Start Loader for CP/M 2.....	D-1
E	A Skeletal Cold Start Loader.....	E-1
F	CP/M Disk Definition Library .....	F-1
G	Blocking and Deblocking Algorithms .....	G-1
H	Glossary .....	H-1
I	CP/M Messages .....	I-1
J	CP/M on the BBC Microcomputer – Technical Notes .....	J-1

## Tables

1-1	Line-editing Control Characters.....	1-12
1-2	CP/M Transient Commands .....	1-14
1-3	Physical Devices.....	1-17
1-4	PIP Parameters.....	1-31
2-1	ED Text Transfer Commands .....	2-3
2-2	Editing Commands .....	2-8
2-3	Line-editing Controls .....	2-9
2-4	Error Message Symbols .....	2-18
2-5	ED Control Characters.....	2-19
2-6	ED Commands .....	2-20
3-1	Reserved Characters .....	3-6
3-2	Arithmetic and Logical Operations .....	3-7
3-3	Assembler Directives.....	3-10
3-4	Jumps, Calls, and Returns .....	3-17
3-5	Immediate Operand Instructions.....	3-19
3-6	Increment and Decrement Instructions.....	3-20

# Table of Contents (continued)

3-7	Data Movement Instructions .....	3-21
3-8	Arithmetic Logic Unit Operations .....	3-22
3-9	Error Codes .....	3-24
3-10	Error Messages .....	3-25
4-1	Line-editing Controls .....	4-2
4-2	DDT Commands .....	4-2
4-3	CPU Registers .....	4-11
5-1	CP/M Filetypes .....	5-7
5-2	File Control Block Fields .....	5-9
5-3	Edit Control Characters .....	5-16
6-1	Standard Memory Size Values .....	6-3
6-2	Common Values for CP/M Systems .....	6-8
6-3	CP/M Disk Sector Allocation .....	6-14
6-4	IOBYTE Field Values .....	6-18
6-5	BIOS Entry Points .....	6-20
6-6	Reserved Locations in Page Zero .....	6-26
6-7	Disk Parameter Headers .....	6-28
6-8	BSH and BLM Values .....	6-31
6-9	EXM Values .....	6-32
6-10	BLS Tabulation .....	6-33

## Figures

2-1	Overall ED Operation .....	2-2
2-2	Memory Buffer Organization .....	2-3
2-3	Logical Organization of Memory Buffer .....	2-5
5-1	CP/M Memory Organization .....	5-2
5-2	File Control Block Format .....	5-8
6-1	IOBYTE Fields .....	6-18
6-2	Disk Parameter Header Format .....	6-28
6-3	Disk Parameter Header Table .....	6-29
6-4	Disk Parameter Block Format .....	6-30
6-5	AL0 and AL1 .....	6-32

## **Table of Contents (continued)**

### **Listings**

6-1	GETSYS Program .....	6-12
6-2	BIOS Entry Points .....	6-16

# Section 1

## CP/M Features and Facilities

### 1.1 Introduction

CP/M® is a monitor control program for microcomputer system development that uses floppy disks or Winchester hard disks for backup storage. Using a computer system based on the Intel® 8080 microcomputer, CP/M provides an environment for program construction, storage, and editing, along with assembly and program checkout facilities. CP/M can be easily altered to execute with any computer configuration that uses a Zilog Z80® or an Intel 8080 Central Processing Unit (CPU) and has at least 20K bytes of main memory with up to 16 disk drives. A detailed discussion of the modifications required for any particular hardware environment is given in Section 6. Although the standard Digital Research version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output (I/O) drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of programs can be stored in both source and machine-executable form.

CP/M 2 is a high-performance, single console operating system that uses table-driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All fundamental file restrictions are removed, maintaining upward compatibility from previous versions of release 1.

Features of CP/M 2 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability of expanding to thirty-two megabytes in future releases. The directory size can be field-configured to contain any reasonable number of entries, and each file is optionally tagged with Read-Only and system attributes. Users of CP/M 2 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2 that provide direct access to any of the 65536 records of an eight-megabyte file.

CP/M also supports ED, a powerful context editor, ASM™, an Intel-compatible assembler, and DDT™, debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled with CP/M's Console Command Processor (CCP), the resulting facilities equal or exceed similar large computer facilities.

CP/M is logically divided into several distinct parts:

- BIOS (Basic I/O System), hardware-dependent
- BDOS (Basic Disk Operating System)
- CCP (Console Command Processor)
- TPA (Transient Program Area)

The BIOS provides the primitive operations necessary to access the disk drives and to interface standard peripherals: teletype, CRT, paper tape reader/punch, and user-defined peripherals. You can tailor peripherals for any particular hardware environment by patching this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies that provide fully dynamic file construction while minimizing head movement across the disk during access. The BDOS has entry points that include the following primitive operations, which the program accesses:

- SEARCH looks for a particular disk file by name.
- OPEN opens a file for further operations.
- CLOSE closes a file after processing.
- RENAME changes the name of a particular file.
- READ reads a record from a particular file.
- WRITE writes a record to a particular file.
- SELECT selects a particular disk drive for further operations.

The CCP provides a symbolic interface between your console and the remainder of the CP/M system. The CCP reads the console device and processes commands, which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands that are available in the CCP are listed in Section 1.2.1.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs that are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

Any or all of the CP/M component subsystems can be overlaid by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A bootstrap loader is programmatically accessible whenever the BIOS portion is not overlaid; thus, the user program need only branch to the bootstrap loader at the end of execution and the complete CP/M monitor is reloaded from disk.

The CP/M operating system is partitioned into distinct modules, including the BIOS portion that defines the hardware environment in which CP/M is executing. Thus, the standard system is easily modified to any nonstandard environment by changing the peripheral drivers to handle the custom system.

## 1.2 Functional Description

You interact with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks that are on-line. The standard system addresses up to sixteen different disk drives. These disk drives are labeled A through P. A disk is logged-in if the CCP is currently addressing the disk. To clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol >, indicating that the CCP is ready for another command. Upon initial start-up, the CP/M system is loaded from disk A, and the CCP displays the following message:

CP/M VER x.x

where x.x is the CP/M version number. All CP/M systems are initially set to operate in a 20K memory space, but can be easily reconfigured to fit any memory size on the host system (see Section 1.6.9). Following system sign-on, CP/M automatically logs in disk A, prompts you with the symbol A>, indicating that CP/M is currently addressing disk A, and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

### 1.2.1 General Command Structure

Built-in commands are a part of the CCP program, while transient commands are loaded into the TPA from disk and executed. The following are built-in commands:

- ERA erases specified files.
- DIR lists filenames in the directory.
- REN renames the specified file.
- SAVE saves memory contents in a file.
- TYPE types the contents of a file on the logged disk.

Most of the commands reference a particular file or group of files. The form of a file reference is specified in Section 1.2.2.

### 1.2.2 File References

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references are either unambiguous (ufn) or ambiguous (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference is satisfied by a number of different files.

File references consist of two parts: the primary filename and the filetype. Although the filetype is optional, it usually is generic. For example, the filetype ASM is used to denote that the file is an assembly language source file, while the primary filename distinguishes each particular source file. The two names are separated by a period, as shown in the following example:

```
filename.typ
```

In this example, filename is the primary filename of eight characters or less, and typ is the filetype of no more than three characters. As mentioned above, the name

```
filename
```

is also allowed and is equivalent to a filetype consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the following special characters:

```
< > . , ; : = ? * [ ] % | ( ) / \
```

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol ? can be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the ? symbol matches any character of a filename in the ? position. Thus, the ambiguous reference

```
X?Z.C?M
```

matches the following unambiguous filenames

XYZ.COM

and

X3Z.CAM

The \* wildcard character can also be used in an ambiguous file reference. The \* character replaces all or part of a filename or filetype. Note that

\* , \*

equals the ambiguous file reference

?????????, ???

while

filename,\*

and

\* , t Y P

are abbreviations for

filename, ???

and

?????????, t Y P

respectively. As an example,

A>DIR \* , \*

is interpreted by the CCP as a command to list the names of all disk files in the directory. The following example searches only for a file by the name X.Y:

A>DIR X , Y

Similarly, the command

**A>DIR X?Y.C?M**

causes a search for all unambiguous filenames on the disk that satisfy this ambiguous reference.

The following file references are valid unambiguous file references:

X  
X,Y  
XYZ  
XYZ.COM  
GAMMA  
GAMMA.1

As an added convenience, the programmer can generally specify the disk drive name along with the filename. In this case, the drive name is given as a letter A through P followed by a colon (:). The specified drive is then logged-in before the file operation occurs. Thus, the following are valid file references with disk name prefixes:

A:X,Y  
P:XYZ.COM  
B:XYZ  
B:X,A?M  
C:GAMMA  
C:\*,ASM

All alphabetic lower-case letters in file and drive names are translated to upper-case when they are processed by the CCP.

## 1.3 Switching Disks

The operator can switch the currently logged disk by typing the disk drive name, A through P, followed by a colon when the CCP is waiting for console input. The following sequence of prompts and commands can occur after the CP/M system is loaded from disk A:

CP/M VER 2.2

A>DIR

A: SAMPLE ASM SAMPLE PRN

A>B:

B>DIR \*.ASM

B:DUMP ASM FILES ASM

B>A:

List all files on disk A.

Switch to disk B.

List all ASM files on B.

Switch back to A.

## 1.4 Built-in Commands

The file and device reference forms described can now be used to fully specify the structure of the built-in commands. Assume the following abbreviations in the description below:

ufn	unambiguous file reference
afn	ambiguous file reference

Recall that the CCP always translates lower-case characters to upper-case characters internally. Thus, lower-case alphabetics are treated as if they are upper-case in command names and file references.

### 1.4.1 ERA Command

#### Syntax:

ERA afn

The ERA (erase) command removes files from the currently logged-in disk, for example, the disk name currently prompted by CP/M preceding the >. The files that are erased are those that satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

ERA X,Y

The file named X,Y on the currently logged disk is removed from the disk directory and the space is returned.

ERA X,\*

All files with primary name X are removed from the current disk.

ERA \*,ASM

All files with secondary name ASM are removed from the current disk.

ERA X?Y,C?M

All files on the current disk that satisfy the ambiguous reference X?Y,C?M are deleted.

ERA \*,\*

Erase all files on the current disk. In this case, the CCP prompts the console with the message

ALL FILES (Y/N)?

which requires a Y response before files are actually removed.

ERA b:\*,PRN

All files on drive B that satisfy the ambiguous reference ????????,PRN are deleted, independently of the currently logged disk.

### 1.4.2 DIR Command

#### Syntax:

DIR afn

The DIR (directory) command causes the names of all files that satisfy the ambiguous filename afn to be listed at the console device. As a special case, the command

DIR

lists the files on the currently logged disk (the command DIR is equivalent to the command DIR \*.\*). The following are valid DIR commands:

DIR X.Y

DIR X?Z.C?M

DIR ??,Y

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place:

DIR B:

DIR B:X,Y

DIR B:\*,A?M

If no files on the selected disk satisfy the directory request, the message NO FILE appears at the console.

### 1.4.3 REN Command

#### Syntax:

REN ufn1 = ufn2

The REN (rename) command allows you to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn2). You can also type a left-directed arrow instead of the equal sign if the console supports this graphic character. The following are examples of the REN command:

REN X,Y=Q,R

The file Q,R is changed to X,Y.

REN XYZ.COM=XYZ,XXX

The file XYZ,XXX is changed to XYZ.COM.

The operator precedes either ufn1 or ufn2 (or both) by an optional drive address. If ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to exist on the drive as well. The same drive must be specified in both cases if both ufn1 and ufn2 are preceded by drive names. The following REN commands illustrate this format:

REN A:X.ASM=Y:ASM

The file Y.ASM is changed to X.ASM on drive A.

REN B:ZAP.BAS=ZOT.BAS

The file ZOT.BAS is changed to ZAP.BAS on drive B.

REN B:A.ASM=B:A.BAK

The file A.BAK is renamed to A.ASM on drive B.

If ufn1 is already present, the REN command responds with the error FILE EXISTS and not perform the change. If ufn2 does not exist on the specified disk, the message NO FILE is printed at the console.

### 1.4.4 SAVE Command

#### Syntax:

SAVE n ufn

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal) which is the second page of memory. The SAVE command must specify 2 pages of memory if the user's program occupies the area from 100H through 2FFH. The machine code file can be subsequently loaded and executed. The following are examples of the SAVE command:

SAVE 3X.COM Copies 100H through 3FFH to X.COM .

SAVE 40 Q Copies 100H through 28FFH to Q. Note that 28 is the page count in 28FFH, and that  $28H = 2^* 16 + 8 = 40$  decimal.

SAVE 4 X.Y Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the ufn portion of the command, as shown in the following example:

SAVE 10 B:ZOT.COM Copies 10 pages, 100H through 0AFFH, to the file ZOT.COM on drive B.

### 1.4.5 TYPE Command

#### Syntax:

TYPE ufn

The TYPE command displays the content of the ASCII source file ufn on the currently logged disk at the console device. The following are valid TYPE commands:

TYPE X.Y

TYPE X.PLM

TYPE XXX

The TYPE command expands tabs, CTRL-I characters, assuming tab positions are set at every eighth column. The ufn can also reference a drive name.

TYPE B:X.PRN

The file X.PRN from drive B is displayed.

#### 1.4.6 USER Command

##### Syntax:

USER n

The USER command allows maintenance of separate files in the same directory. In the syntax line, n is an integer value in the range 0 to 15. On cold start, the operator is automatically logged into user area number 0, which is compatible with standard CP/M 1 directories. You can issue the USER command at any time to move to another logical area within the same directory. Drives that are logged-in while addressing one user number are automatically active when the operator moves to another. A user number is simply a prefix that accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start when user 0 is again assumed.

### 1.5 Line Editing and Output Control

The CCP allows certain line-editing functions while typing command lines. The CTRL-key sequences are obtained by pressing the control and letter keys simultaneously. Further, CCP command lines are generally up to 255 characters in length; they are not acted upon until the carriage return key is pressed.

Table 1-1. Line-editing Control Characters

Character	Meaning
CTRL-C	Reboots CP/M system when pressed at start of line.
CTRL-E	Physical end of line; carriage is returned, but line is not sent until the carriage return key is pressed.
CTRL-H	Backspaces one character position.

Table 1-1. (continued)

Character	Meaning
CTRL-J	Terminates current input (line-feed).
CTRL-M	Terminates current input (carriage return).
CTRL-P	Copies all subsequent console output to the currently assigned list device (see Section 1.6.1). Output is sent to the list device and the console device until the next CTRL-P is pressed.
CTRL-R	Retypes current command line; types a clean line following character deletion with rubouts.
CTRL-S	Stops the console output temporarily. Program execution and output continue when you press any character at the console, for example another CTRL-S. This feature stops output on high speed consoles, such as CRTs, in order to view a segment of output before continuing.
CTRL-U	Deletes the entire line typed at the console.
CTRL-X	Same as CTRL-U.
CTRL-Z	Ends input from the console (used in PIP and ED).
rub/del	Deletes and echoes the last character typed at the console.

## 1.6 Transient Commands

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands for execution under the CCP are below. Additional functions are easily defined by the user (see Section 1.6.3).

**Table 1-2. CP/M Transient Commands**

<i>Command</i>	<i>Function</i>
STAT	Lists the number of bytes of storage remaining on the currently logged disk, provides statistical information about particular files, and displays or alters device assignment.
ASM	Loads the CP/M assembler and assembles the specified program from disk.
LOAD	Loads the file in Intel HEX machine code format and produces a file in machine executable form which can be loaded into the TPA. This loaded program becomes a new command under the CCP.
DDT	Loads the CP/M debugger into TPA and starts execution.
PIP	Loads the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.
ED	Loads and executes the CP/M text editor program.
SYSGEN	Creates a new CP/M system disk.
SUBMIT	Submits a file of commands for batch processing.
DUMP	Dumps the contents of a file in hex.
MOVCPM	Regenerates the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands are easily defined by the user. For convenience, the transient command can be preceded by a drive name which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

B : STAT

causes CP/M to temporarily log in drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

### 1.6.1 STAT Command

#### Syntax:

STAT

STAT "command line"

The STAT command provides general statistical information about file storage and device assignment. Special forms of the command line allow the current device assignment to be examined and altered. The various command lines that can be specified are shown with an explanation of each form to the right.

STAT      If you type an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints one of the following messages:

d : R/W , SPACE: nnnK

d : R/O , SPACE: nnnK

for each active drive d:, where R/W indicates the drive can be read or written, and R/O indicates the drive is Read-Only (a drive becomes R/O by explicitly setting it to Read-Only, as shown below, or by inadvertently changing disks without performing a warm start). The space remaining on the disk in drive d: is given in kilobytes by nnn.

STAT d:      If a drive name is given, then the drive is selected before the storage is computed. Thus, the command STAT B: could be issued while logged into drive A, resulting in the message

BYTES REMAINING ON B: nnnK

**STAT afn** The command line can also specify a set of files to be scanned by STAT. The files that satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading:

```
RECS BYTES EXT D:FILENAME.TYP  
rrrr bbbK ee d:filename.typ
```

where rrrr is the number of 128-byte records allocated to the file, bbb is the number of kilobytes allocated to the file ( $bbb = rrrr * 128 / 1024$ ), ee is the number of 16K extensions ( $ee = bbb / 16$ ), d is the drive name containing the file (A...P), filename is the eight-character primary filename, and typ is the three-character filetype. After listing the individual files, the storage usage is summarized.

**STAT d:afn** The drive name can be given ahead of the afn. The specified drive is first selected, and the form STAT afn is executed.

**STAT d:=R/O** This form sets the drive given by d to Read-Only, remaining in effect until the next warm or cold start takes place. When a disk is Read-Only, the message

```
BDOS ERR ON d: Read-Only
```

appears if there is an attempt to write to the Read-Only disk. CP/M waits until a key is pressed before performing an automatic warm start, at which time the disk becomes R/W.

The STAT command allows you to control the physical-to-logical device assignment. See the IOBYTE function described in Sections 5 and 6. There are four logical peripheral devices that are, at any particular instant, each assigned one of several physical peripheral devices. The following is a list of the four logical devices:

- **CON:** is the system console device, used by CCP for communication with the operator.
- **RDR:** is the paper tape reader device.
- **PUN:** is the paper tape punch device.
- **LST:** is the output list device.

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could actually be a high speed reader, teletype reader, or cassette tape. To allow some flexibility in device naming and assignment, several physical devices are defined in Table 1-3.

Table 1-3. Physical Devices

<i>Device</i>	<i>Meaning</i>
TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR:, output goes to current LST: device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

It is emphasized that the physical device names might not actually correspond to devices that the names imply. That is, you can implement the PTP: device as a cassette write operation. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The command,

**STAT VAL:**

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk d:$R/O
Set Indicator: filename,typ $R/O $R/W $SYS $DIR
Disk Status: DSK: d:DSK
Iobyte Assign:
```

which gives an instant summary of the possible STAT commands and shows the permissible logical-to-physical device assignments:

```
CON:=TTY:CRT:BAT:UC1:
RDR:=TTY:PTR:UR1:UR2:
PUN:=TTY:PTP:UP1:UP2:
LST:=TTY:CRT:LPT:UL1:
```

The logical device to the left takes any of the four physical assignments shown to the right. The current logical-to-physical mapping is displayed by typing the command:

**STAT DEV:**

This command produces a list of each logical device to the left and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON:=CRT:
RDR:=UR1:
PUN:=PTP:
LST:=TTY:
```

The current logical-to-physical device assignment is changed by typing a STAT command of the form:

STAT ld1 = pd1, ld2 = pd2, . . . , ldn = pdn

where ld1 through ldn are logical device names and pd1 through pdn are compatible physical device names. For example, ld1 and pd1 appear on the same line in the VAL: command shown above. The following example shows valid STAT commands that change the current logical-to-physical device assignments:

```
STAT CON:=CRT:  
STAT PUN:=TTY:, LST:=LPT:, RDR:=TTY:
```

The command form,

STAT d:filename.typ \$\$

where d: is an optional drive name and filename.typ is an unambiguous or ambiguous filename, produces the following output display format:

Size	Recs	Bytes	Ext Acc
48	48	6K	1 R/O A:ED.COM
55	55	12K	1 R/O (A:PIP.COM)
65536	128	16K	2 R/W A:X.DAT

where the \$\$ parameter causes the Size field to be displayed. Without the \$\$, the Size field is skipped, but the remaining fields are displayed. The Size field lists the virtual file size in records, while the Recs field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The Bytes field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time; thus, the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent. Each of these extents, however, can contain unallocated holes even though they are added into the record count.

The Ext field counts the number of physical extents allocated to the file. The Ext count corresponds to the number of directory entries given to the file. Depending on allocation size, there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry. In a special case, there are actually 256K bytes that can be directly addressed by a physical extent.

The Acc field gives the R/O or R/W file indicator, which you can change using the commands shown. The four command forms,

```
STAT d:filename.typ $R/O  
STAT d:filename.typ $R/W  
STAT d:filename.typ $SYS  
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file, or set of files, in a Read-Only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent Read-Write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The filename.typ may be ambiguous or unambiguous, but files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by d: is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file produce the following BDOS message at your screen:

**BDOS Err on d: File R/O**

lists the drive characteristics of the disk named by d: that is in the range A:, B:,...,P:. The drive characteristics are listed in the following format:

d: Drive Characteristics

65536: 128 Byte Record Capacity  
8192: Kilobyte Drive Capacity  
128: 32 Byte Directory Entries  
0: Checked Directory Entries  
1024: Records/Extent  
128: Records/Block  
58: Sectors/Track  
2: Reserved Tracks

where d: is the selected drive, followed by the total record capacity (65536 is an eight-megabyte drive), followed by the total capacity listed in kilobytes. The directory size is listed next, followed by the checked entries. The number of checked entries is usually identical to the directory size for removable media, because this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, because the media are not changed without at least a cold or warm start.

The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or 128K in the previous example). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks.

For logical drives that share the same physical disk, the number of reserved tracks can be quite large because this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

**STAT DSK:**

produces a drive characteristics table for all currently active drives. The final STAT command form is

**STAT USR:**

which produces a list of the user numbers that have files on the currently addressed disk. The display format is

```
Active User: 0  
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In this case, the active user number is 0 (default at cold start) with three user numbers that have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging in with USER 1 or USER 3 commands, followed by a DIR command at the CCP level.

### 1.6.2 ASM Command

**Syntax:**

```
ASM ufn
```

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements, where the filetype is assumed to be ASM and is not specified. The following ASM commands are valid:

```
ASM X  
ASM GAMMA
```

The two-pass assembler is automatically executed. Assembly errors that occur during the second pass are printed at the console.

The assembler produces a file:

X.PRN

where X is the primary name specified in the ASM command. The PRN file contains a listing of the source program with embedded tab characters if present in the source program, along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file is listed at the console using the TYPE command, or sent to a peripheral device using PIP (see Section 1.6.4). Note that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns; for example, program addresses and hexadecimal machine code. The PRN file serves as a backup for the original source file. If the source file is accidentally removed or destroyed, the PRN file can be edited by removing the leftmost 16 characters of each line (see Section 2). This is done by issuing a single editor macro command. The resulting file is identical to the original source file and can be renamed from PRN to ASM for subsequent editing and assembly. The file

X.HEX

is also produced, which contains 8080 machine language in Intel HEX format suitable for subsequent loading and execution (see Section 1.6.3). For complete details of CP/M's assembly language program, see Section 3.

The source file for assembly is taken from an alternate disk by prefixing the assembly language filename by a disk drive name. The command

ASM B:ALPHA

loads the assembler from the currently logged drive and processes the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

### 1.6.3 LOAD Command

#### Syntax:

LOAD ufn

The LOAD command reads the file ufn, which is assumed to contain HEX format machine code, and produces a memory image file that can subsequently be executed. The filename ufn is assumed to be of the form:

X.HEX

and only the filename X need be specified in the command. The LOAD command creates a file named

X.COM

that marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the filename X immediately after the prompting character > printed by the CCP.

Generally, the CCP reads the filename X following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

X.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by typing the primary name. This way, you can invent new commands in the CCP. Initialized disks contain the transient commands as COM files, which are optionally deleted. The operation takes place on an alternate drive if the filename is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates on drive B after execution begins.

**Note:** the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) that begin at 100H of the TPA. The addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard COM files that operate in the TPA. Programs that occupy regions of memory other than the TPA are loaded under DDT.

#### 1.6.4 PIP

##### Syntax:

PIP

PIP destination = source#1, source#2, . . . , source #n

PIP is the CP/M Peripheral Interchange Program that implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms:

PIP

PIP command line

In both cases PIP is loaded into the TPA and executed. In the first form, PIP reads command lines directly from the console, prompted with the \* character, until an empty command line is typed (for example, a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below.

In the second form, the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, . . . , source#n

where destination is the file or peripheral device to receive the data, and source#1, . . . , source#n is a series of one or more files or devices that are copied from left to right to the destination.

When multiple files are given in the command line (for example,  $n > 1$ ), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (CTRL-Z) at the end of each file (see the O parameter to override this assumption). Lower-case ASCII alphabetics are internally translated to upper-case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters. CTRL-E can be used to force a physical carriage return for lines that exceed the console width.

The destination and source elements are unambiguous references to CP/M source files with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A: through P:) that defines the particular drive where the file can be obtained or stored. When the drive name is not included, the currently logged disk is assumed. The destination file can also appear as one or more of the source files; in which case the source file is not altered until the entire concatenation is complete. If it already exists, the destination file is removed if the command line is properly formed. It is not removed if an error condition arises. The following command lines, with explanations to the right, are valid as input to PIP:

X = Y	Copies to file X from file Y, where X and Y are unambiguous filenames; Y remains unchanged.
X = Y , Z	Concatenates files Y and Z and copies to file X, with Y and Z unchanged.
X . ASM = Y . ASM , Z . ASM	Creates the file X . ASM from the concatenation of the Y and Z . ASM files.
NEW . ZOT = B : OLD . ZAP	Moves a copy of OLD . ZAP from drive B to the currently logged disk; names the file NEW . ZOT.
B : A . U = B : B . V , A : C . W , D . X	Concatenates file B . V from drive B with C . W from drive A and D . X from the logged disk; creates the file A . U on drive B.

For convenience, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated PIP forms are

PIP d:=afn  
PIP d<sub>1</sub>=d<sub>2</sub>:afn  
PIP ufn=d<sub>2</sub>:  
PIP d<sub>1</sub>:ufn=d<sub>2</sub>:

The first form copies all files from the currently logged disk that satisfy the afn to the same files on drive d, where d = A..P. The second form is equivalent to the first, where the source for the copy is drive d<sub>2</sub>, where d<sub>2</sub> = A..P. The third form is equivalent to the command PIP d<sub>1</sub>:ufn=d<sub>2</sub>:ufn which copies the file given by ufn from drive d<sub>2</sub> to the file ufn on drive d<sub>1</sub>:. The fourth form is equivalent to the third, where the source disk is explicitly given by d<sub>2</sub>..

The source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn that satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed after successful completion of the copy and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

B=*,COM	Copies all files that have the secondary name COM to drive B from the current drive.
A:=B:ZAP,*	Copies all files that have the primary name ZAP to drive A from drive B.
ZAP,ASM=B:	Same as ZAP,ASM=B:ZAP,ASM
B:ZOT,COM=A:	Same as B:ZOT,COM=A:ZOT,COM
B:=GAMMA,BAS	Same as B:GAMMA,BAS=GAMMA,BAS
B:=A:GAMMA,BAS	Same as B:GAMMA,BAS=A:GAMMA,BAS

PIP allows reference to physical and logical devices that are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The following is a list of logical devices given in the STAT command

CON: (console)  
RDR: (reader)  
PUN: (punch)  
LST: (list)

while the physical devices are

TTY: (console , reader, punch, or list)  
CRT: (console, or list), UC1: (console)  
PTR: (reader), UR1: (reader), UR2: (reader)  
PTP: (punch), UP1: (punch), UP2: (punch)  
LPT: (list), UL1: (list)

The BAT: physical device is not included, because this assignment is used only to indicate that the RDR: and LST: devices are used for console input/output.

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and are easily altered for any particular I/O system. The current physical device mapping is defined by IOBYTE; see Section 6 for a discussion of this function. The destination device must be capable of receiving data, for example, data cannot be sent to the punch, and the source devices must be capable of generating data, for example, the LST: device cannot be read.

The following list describes additional device names that can be used in PIP commands.

- NUL: sends 40 nulls (ASCII 0s) to the device. This can be issued at the end of punched output.
- EOF: sends a CP/M end-of-file (ASCII CTRL-Z) to the destination device (sent automatically at the end of all ASCII data transfers through PIP).
- INP: is a special PIP input source that can be patched into the PIP program. PIP gets the input data character-by-character, by CALLing location 103H, with data returned in location 109H (parity bit must be zero).

- OUT: is a special PIP output destination that can be patched into the PIP program. PIP CALLS location 106H with data in register C for each character to transmit. Note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT (see Section 4).
- PRN: is the same as LST:, except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines with an initial eject (same as using PIP options [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (CTRL-Z for ASCII files, and end-of-data for non-ASCII disk files). Data from each device or file are concatenated from left to right until the last data source has been read.

The destination device or file is written using the data from the source files, and an end-of-file character, CTRL-Z, is appended to the result for ASCII files. If the destination is a disk file, a temporary file is created (\$\$\$ secondary name) that is changed to the actual filename only on successful completion of the copy. Files with the extension COM are always assumed to be non-ASCII.

The copy operation can be aborted at any time by pressing any key on the keyboard. PIP responds with the message ABORTED to indicate that the operation has not been completed. If any operation is aborted, or if an error occurs during processing, PIP removes any pending commands that were set up while using the SUBMIT command.

PIP performs a special function if the destination is a disk file with type HEX (an Intel hex-formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records.

When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. Usually, you can open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the reread, a single carriage return is typed at the console, and PIP attempts another read. If the tape position cannot be properly read, continue the read by typing a return following the error message, and enter the record manually with the ED program after the disk file is constructed.

PIP allows the end-of-file to be entered from the console if the source file is an RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If CTRL-Z is typed at the keyboard, the read operation is terminated normally.

The following are valid PIP commands:

**PIP LST:=X, PRN**

Copies X.PRN to the LST device and terminates the PIP program.

**PIP**

Starts PIP for a sequence of commands. PIP prompts with \*.

**\*CON:=X,ASM,Y,ASM,Z,ASM**

Concatenates three ASM files and copies to the CON device.

**\*X,HEX=CON:,Y,HEX,PTR:**

Creates a HEX file by reading the CON until a CTRL-Z is typed, followed by data from Y.HEX and PTR until a CTRL-Z is encountered.

**PIP PUN:=NUL:,X,ASM,EOF:,NUL:**

Sends 40 nulls to the punch device; copies the X.ASM file to the punch, followed by an end-of-file, CTRL-Z, and 40 more null characters.

**(carriage return)**

A single carriage return stops PIP.

You can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). Table 1-4 describes valid PIP parameters.

Table 1-4. PIP Parameters

Parameter	Meaning
B	Blocks mode transfer. Data are buffered by PIP until an ASCII x-off character, CTRL-S, is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data that can be buffered depends on the memory size of the host system. PIP issues an error message if the buffers overflow.
Dn	Deletes characters that extend past column n in the transfer of data to the destination from the character source. This parameter is generally used to truncate long lines that are sent to a narrow printer or console device.
E	Echoes all transfer operations to the console as they are being performed.
F	Filters form-feeds from the file. All embedded form-feeds are removed. The P parameter can be used simultaneously to insert new form-feeds.
Gn	Gets file from user number n (n in the range 0-15).
H	Transfers HEX data. All data are checked for proper Intel hex file format. Nonessential characters between hex records are removed during the copy operation. The console is prompted for corrective action in case errors occur.
I	Ignores :00 records in the transfer of Intel hex format file. The I parameter automatically sets the H parameter.
L	Translates upper-case alphabetics to lower-case.
N	Adds line numbers to each line transferred to the destination, starting at one and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, leading zeroes are included and a tab is inserted following the number. The tab is expanded if T is set.

Table 1-4. (continued)

<i>Parameter</i>	<i>Meaning</i>
O	Transfers non-ASCII object files. The normal CP/M end-of-file is ignored.
Pn	Includes page ejects at every n lines with an initial page eject. If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form-feed suppression takes place before the new page ejects are inserted.
Qs↑Z	Quits copying from the source device or file when the string s, terminated by CTRL-Z, is encountered.
R	Reads system files.
Ss↑Z	Start copying from the source device when the string s, terminated by CTRL-Z, is encountered. The S and Q parameters can be used to abstract a particular section of a file, such as a subroutine. The start and quit strings are always included in the copy operation.
	If you specify a command line after the PIP command keyword, the CCP translates strings following the S and Q parameters to upper-case. If you do not specify a command line, PIP does not perform the automatic upper-case translation.
Tn	Expands tabs, CTRL-I characters, to every nth column during the transfer of characters to the destination from the source.
U	Translates lower-case alphabets to upper-case during the copy operation.
V	Verifies that data have been copied correctly by rereading after the write operation (the destination must be a disk file).
W	Writes over R/O files without console interrogation.
Z	Zeros the parity bit on input for each ASCII character.

The following examples show valid PIP commands that specify parameters in the file transfer.

**PIP X.ASM=B:[v]**

Copies X.ASM from drive B to the current drive and verifies that the data were properly copied.

**PIP LPT:=X.ASM[n8u]**

Copies X.ASM to the LPT: device; numbers each line, expands tabs to every eighth column, and translates lower-case alphabetics to upper-case.

**PIP PUN:=X.HEX[i],Y.ZOT[h]**

First copies X.HEX to the PUN: device and ignores the trailing :00 record in X.HEX; continues the transfer of data by reading Y.ZOT, which contains HEX records, including any :00 records it contains.

**PIP X.LIB=Y.ASM[s]SUBR1:↑zqJMP L3↑z]**

Copies from the file Y.ASM into the file X.LIB. The command starts the copy when the string SUBR1: has been found, and quits copying after the string JMP L3 is encountered.

**PIP PRN:=X.ASM[p50]**

Sends X.ASM to the LST: device with line numbers, expands tabs to every eighth column, and ejects pages at every 50th line. The assumed parameter list for a PRN file is nt8p60; p50 overrides the default value.

Under normal operation, PIP does not overwrite a file that is set to a permanent R/O status. If an attempt is made to overwrite an R/O file, the following prompt appears:

**DESTINATION FILE IS R/O, DELETE (Y/N)?**

If you type Y, the file is overwritten. Otherwise, the following response appears:

**\*\* NOT DELETED \*\***

The file transfer is skipped, and PIP continues with the next operation in sequence. To avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter, as shown in this example:

```
PIP A:=B:*.COM[W]
```

The W parameter copies all nonsystem files to the A drive from the B drive and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as in this example:

```
PIP A.DAT=B.DAT,F:NEW.DAT,G:OLD.DAT[W]
```

Files with the system attribute can be included in PIP transfers if the R parameter is included; otherwise, system files are not recognized. For example, the command line:

```
PIP ED.COM=B:ED.COM[R]
```

reads the ED.COM file from the B drive, even if it has been marked as an R/O and system file. The system file attributes are copied, if present.

Downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by user 0. If compatibility is required with nonstandard, for example, double-density versions of 1.4, it might be necessary to select 1.4 compatibility mode when constructing the internal disk parameter block. See Section 6 and refer to Section 6.10, which describes BIOS differences.

**Note:** to copy files into another user area, PIP.COM must be located in that user area. Use the following procedure to make a copy of PIP.COM in another user area.

USER 0	Log in user 0.
DDT PIP.COM (note PIP size s)	Load PIP to memory.
GO	Return to CCP.
USER 3	Log in user 3.
SAVEs PIP.COM	

In this procedure, s is the integral number of memory pages, 256-byte segments, occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the NEXT display. If, for example, the next

available address is 1D00, then PIP.COM requires 1C hexadecimal pages, or 1 times 16 + 12 = 28 pages, and the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can be copied to another disk belonging to the same user number through normal PIP transfers.

### 1.6.5 ED Command

#### Syntax:

ED ufn

The ED program is the CP/M system context editor that allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given in Section 2. ED allows the operator to create and operate upon source files that are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage return/line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory) that is defined by the number of characters typed between carriage returns.

The ED program has a number of commands for character string searching, replacement, and insertion that are useful for creating and correcting programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 20K CP/M system), the file size that can be edited is not limited, since data are easily paged through this work area.

If it does not exist, ED creates the specified source file and opens the file for access. If the source file does exist, the programmer appends data for editing (see the A command). The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context, allowing easy access to particular portions of a large file (see the N command).

If you type the following command line:

ED X.ASM

the ED program creates an intermediate work file with the name

X.\$\$\$

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original unedited file, and the X.ASM file contains

the newly edited file. The operator can always return to the previous version of a file by removing the most recent version and renaming the previous version. If the current X.ASM file has been improperly edited, the following sequence of commands reclaim the back-up file.

DIR X.*	Checks to see that BAK file is available.
ERA X.ASM	Erases most recent version.
REN X.ASM=X.BAK	Renames the BAK file to ASM.

You can abort the edit at any point (reboot, power failure, CTRL-C, or CTRL-Q command) without destroying the original file. In this case, the BAK file is not created and the original file is always intact.

The ED program allows the user to edit the source on one disk and create the back-up file on another disk. This form of the ED command is

ED ufn d:

where ufn is the name of the file to edit on the currently logged disk and d is the name of an alternate drive. The ED program reads and processes the source file and writes the new file to drive d using the name ufn. After processing, the original file becomes the back-up file. If the operator is addressing disk A, the following command is valid.

ED X.ASM b:

This edits the file X.ASM on drive A, creating the new file X.\$\$\$ on drive B. After a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.\$\$\$ is renamed to B:X.ASM. For convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file named B:X.ASM exists before the editing begins, the following message appears on the screen:

FILE EXISTS

This message is a precaution against accidentally destroying a source file. You should first erase the existing file and then restart the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source filename by a drive name. The following are examples of valid edit requests:

**ED A:X.ASM** Edits the file X.ASM on drive A, with new file and back-up on drive A.

**ED B:X.ASM A:** Edits the file X.ASM on drive B to the temporary file X.\$\$\$ on drive A. After editing, this command changes X.ASM on drive B to X.BAK and changes X.\$\$\$ on drive A to X.ASM

### 1.6.6 SYSGEN Command

#### Syntax:

**SYSGEN**

The SYSGEN transient command allows generation of an initialized disk containing the CP/M operating system. The SYSGEN program prompts the console for commands by interacting as shown.

**SYSGEN<cr>**

Initiates the SYSGEN program.

**SYSGEN VERSION x.x**

SYSGEN sign-on message.

**SOURCE DRIVE NAME  
(OR RETURN TO SKIP)**

Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system, usually A. If a copy of CP/M already exists in memory due to a MOVCPM command, press only a carriage return. Typing a drive name d causes the response:

**SOURCE ON d THEN TYPE RETURN**

Place a disk containing the CP/M operating system on drive d (d is one of A, B, C, or D). Answer by pressing a carriage return when ready.

**FUNCTION COMPLETE**

System is copied to memory. SYSGEN then prompts with the following:

**DESTINATION DRIVE NAME  
(OR RETURN TO REBOOT)**

If a disk is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, press a carriage return and the system reboots from drive A. Typing drive name d causes SYSGEN to prompt with the following message:

**DESTINATION ON d  
THEN TYPE RETURN**

Place new disk into drive d; press return when ready.

**FUNCTION COMPLETE**

New disk is initialized in drive d.

The DESTINATION prompt is repeated until a single carriage return is pressed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new disk contains the operating system, and only the built-in commands are available. An IBM-compatible disk appears to CP/M as a disk with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M disk to the newly constructed disk using the PIP transient.

You can copy all files from an existing disk by typing the following PIP command:

**PIP B:=A:\*,\* [v]**

This command copies all files from disk drive A to disk drive B and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

Note that a SYSGEN does not destroy the files that already exist on a disk; it only constructs a new operating system. If a disk is being used only on drives B through P and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place.

### 1.6.7 SUBMIT Command

#### Syntax:

```
SUBMIT ufn parm#1 . . . parm#n
```

The SUBMIT command allows CP/M commands to be batched for automatic processing. The ufn given in the SUBMIT command must be the filename of a file that exists on the currently logged disk, with an assumed file type of SUB. The SUB file contains CP/M prototype commands with possible parameter substitution. The actual parameters parm#1 . . . parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed \$ parameters of the form:

```
$1 $2 $3 . . . $n
```

corresponding to the number of actual parameters that will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 . . . parm#n are paired with the formal parameters \$1 . . . \$n in the prototype commands. If the numbers of formal and actual parameters do not correspond, the SUBMIT function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

**\$\$\$ . SUB**

on the logged disk. When the system reboots, at the termination of the SUBMIT, this command file is read by the CCP as a source of input rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. You can abort command processing at any time by pressing the rubout key when the command is read and echoed. In this case, the \$\$\$ . SUB file is removed and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs that execute under CP/M can abort processing of command files when error conditions occur by erasing any existing \$\$\$ . SUB file.

To introduce dollar signs into a SUBMIT file, you can type a \$\$ which reduces to a single \$ within the command file. An up arrow, ↑ , precedes an alphabetic character s, which produces a single CTRL-X character within the file.

The last command in a SUB file can initiate another SUB file, allowing chained batch commands:

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1,*
ERA *,BAK
PIP $2:=$1,PRN
ERA $1,PRN
```

then, you issue the following command:

```
SUBMIT ASMBL X PRN
```

The SUBMIT program reads the ASMBL.SUB file, substituting X: for all occurrences of \$1 and PRN for all occurrences of \$2. This results in a \$\$\$.SUB file containing the commands:

```
ASM X
DIR X,*
ERA *,BAK
PIP PRN:=X,PRN
ERA X,PRN
```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file on an alternate drive by preceding the filename by a drive name. Submitted files are only acted upon when they appear on drive A. Thus, it is possible to create a submitted file on drive B that is executed at a later time when inserted in drive A.

An additional utility program called XSUB extends the power of the SUBMIT facility to include line input to programs as well as the CCP. The XSUB command is included as the first line of the SUBMIT file. When it is executed, XSUB self-relocates directly below the CCP. All subsequent SUBMIT command lines are processed by XSUB so that programs that read buffered console input, BDOS Function 10, receive their input directly from the SUBMIT file. For example, the file SAVER.COM can contain the following SUBMIT lines:

```
XSUB  
DDT  
1$1.COM  
R  
GO  
SAVE 1 $2.COM
```

a subsequent SUBMIT command, such as

```
A>SUBMIT SAVER PIP Y
```

substitutes PIP for \$1 and Y for \$2 in the command stream. The XSUB program loads, followed by DDT, which is sent to the command lines PIP.COM, R, and GO, thus returning to the CCP. The final command SAVE 1 Y.COM is processed by the CCP.

The XSUB program remains in memory and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent SUBMIT command streams do not require the XSUB, unless an intervening cold start occurs. Note that XSUB must be loaded after the optional CP/M DESPOOL utility, if both are to run simultaneously.

### 1.6.8 DUMP Command

Syntax:

```
DUMP ufn
```

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pressing the rubout key during printout. The source listing of the DUMP program is given in Section 5 as an example of a program written for the CP/M environment.

### 1.6.9 MOVCPM Command

#### Syntax:

MOVCPM

The MOVCPM program allows you to reconfigure the CP/M system for any particular memory size. Two optional parameters can be used to indicate the desired size of the new system and the disposition of the new system at program termination. If the first parameter is omitted or an \* is given, the MOVCPM program reconfigures the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if \* is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The following is a list of MOVCPM command forms:

MOVCPM

Relocates and executes CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). On completion of the relocation, the new system is executed but not permanently recorded on the disk. The system that is constructed contains a BIOS for the Intel MDS 800.

MOVCPM n

Creates a relocated CP/M system for management of an n kilobyte system (n must be in the range of 20 to 64), and executes the system as described.

MOVCPM \* \*

Constructs a relocated memory image for the current memory configuration, but leaves the memory image in memory in preparation for a SYSGEN operation.

MOVCPM n \*

Constructs a relocated memory image for an n kilobyte memory system, and leaves the memory image in preparation for a SYSGEN operation.

For example, the command,

**MOVCPM \* \***

constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

**READY FOR 'SYSGEN' OR  
'SAVE 34 CPMxx.COM'**

appears at the console upon completion, where xx is the current memory size in kilobytes. You can then type the following sequence:

**SYSGEN**

This starts the system generation.

**SOURCE DRIVE NAME  
(OR RETURN TO SKIP)**

Respond with a carriage return to skip the CP/M read operation, because the system is already in memory as a result of the previous MOVCPM operation.

**DESTINATION DRIVE NAME  
(OR RETURN TO REBOOT)**

Respond with B to write new system to the disk in drive B. SYSGEN prompts with the following message:

**DESTINATION ON B,  
THEN TYPE RETURN**

Place the new disk on drive B and press the RETURN key when ready.

If you respond with A rather than B above, the system is written to drive A rather than B. SYSGEN continues to print this prompt:

**DESTINATION DRIVE NAME (OR RETURN TO REBOOT)**

until you respond with a single carriage return, which stops the SYSGEN program with a system reboot.

You can then go through the reboot process with the old or new disk. Instead of performing the SYSGEN operation, you can type a command of the form:

SAVE 34 CPMxx.COM

at the completion of the MOVCPM function, where xx is the value indicated in the SYSGEN message. The CP/M memory image on the currently logged disk is in a form that can be patched. This is necessary when operating in a nonstandard environment where the BIOS must be altered for a particular peripheral device configuration, as described in Section 6.

The following are valid MOVCPM commands:

**MOVCPM 48** Constructs a 48K version of CP/M and starts execution.

**MOVCPM 48\*** Constructs a 48K version of CP/M in preparation for permanent recording; the response is

READY FOR 'SYSGEN' OR  
'SAVE 34 CPM48.COM'

**MOVCPM \*\*** Constructs a maximum memory version of CP/M and starts execution.

The newly created system is serialized with the number attached to the original disk and is subject to the conditions of the Digital Research Software Licensing Agreement.

## 1.7 BDOS Error Messages

There are three error situations that the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

BDOS ERR ON d: error

where d is the drive name and error is one of the three error messages:

BAD SECTOR  
SELECT  
READ ONLY

The **BAD SECTOR** message indicates that the disk controller electronics has detected an error condition in reading or writing the disk. This condition is generally caused by a malfunctioning disk controller or an extremely worn disk. If you find that CP/M reports this error more than once a month, the state of the controller electronics and the condition of the media should be checked.

You can also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers claim to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of ones following the data CRC byte, which is not required in the IBM format. As a result, disks generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturers' equipment produce the **BAD SECTOR** message when read by the MDS. To recover from this condition, press a CTRL-C to reboot (the safest course), or a return, which ignores the bad sector in the file operation.

**Note:** pressing a return might destroy disk integrity if the operation is a directory write. Be sure you have adequate back-ups in this case.

The **SELECT** error occurs when there is an attempt to address a drive beyond the range supported by the BIOS. In this case, the value of d in the error message gives the selected drive. The system reboots following any input from the console.

The **READ ONLY** message occurs when there is an attempt to write to a disk or file that has been designated as Read-Only in a **STAT** command or has been set to Read-Only by the BDOS. Reboot CP/M by using the warm start procedure, CTRL-C, or by performing a cold start whenever the disks are changed. If a changed disk is to be read but not written, BDOS allows the disk to be changed without the warm or cold start, but internally marks the drive as Read-Only. The status of the drive is subsequently changed to Read-Write if a warm or cold start occurs. On issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

## 1.8 Operation of CP/M on the MDS

This section gives operating procedures for using CP/M on the Intel MDS micro-computer development system. Basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as the Intel ISIS operating system. The disk drives are labeled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system disk is inserted into drive 0, and the BOOT and RESET switches are pressed in sequence. The interrupt 2 light should go on at this point. The space bar is then pressed on the system console, and the light should go out. If it does not, the user should check connections and baud rates. The BOOT switch is turned off, and the CP/M sign-on message should appear at the selected console device, followed by the A> system prompt. You can then issue the various resident and transient commands.

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch, which generates an RST 7, except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Do not remove a disk and replace it with another without rebooting the system (cold or warm start) unless the inserted disk is Read-Only.

As a result of hardware hang-ups or malfunctions, CP/M might print the following message:

**BDDOS ERR ON d: BAD SECTOR**

where d is the drive that has a permanent error. This error can occur when drive doors are opened and closed randomly, followed by disk operations, or can be caused by a disk, drive, or controller failure. You can optionally elect to ignore the error by pressing a single return at the console. The error might produce a bad data record, requiring reinitialization of up to 128 bytes of data. You can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the disks before turning the power off to avoid random transients that often make their way to the drive electronics.

You should use IBM-compatible disks rather than disks that have previously been used with any ISIS version. In particular, the ISIS FORMAT operation produces non-standard sector numbering throughout the disk. This nonstandard numbering seriously degrades the performance of CP/M, and causes CP/M to operate noticeably slower than the distribution version. If it becomes necessary to reformat a disk, which should not be the case for standard disks, a program can be written under CP/M that causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

Generally, IBM-compatible 8-inch disks do not need to be formatted. However, 5 1/4-inch disks need to be formatted.

*End of Section 1*

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

ג

# Section 2

## The CP/M Editor

### 2.1 Introduction to ED

ED is the context editor for CP/M, and is used to create and alter CP/M source files. To start ED, type a command of the following form:

ED filename

or

ED filename.typ

Generally, ED reads segments of the source file given by filename or filename.typ into the central memory, where you edit the file and it is subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 2-1.

#### 2.1.1 ED Operation

ED operates upon the source file, shown in Figure 2-1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered. The number of lines that can be maintained in the memory buffer varies with the line length, but has a total capacity of about 5000 characters in a 20K CP/M system.

Edited text material is written into a temporary work file under your command. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from x.y to x.BAK so that the most recent edited source file can be reclaimed if necessary. See the CP/M commands ERASE and RENAME. The temporary file is then changed from x.\$\$\$ to x.y, which becomes the resulting edited file.

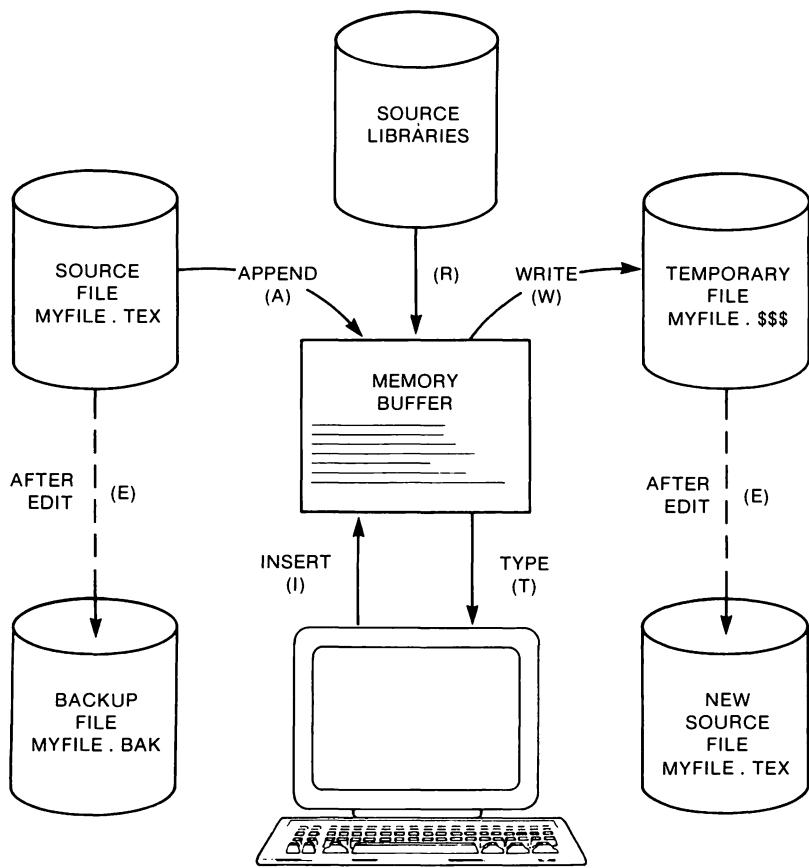


Figure 2-1. Overall ED Operation

The memory buffer is logically between the source file and working file, as shown in Figure 2-2.

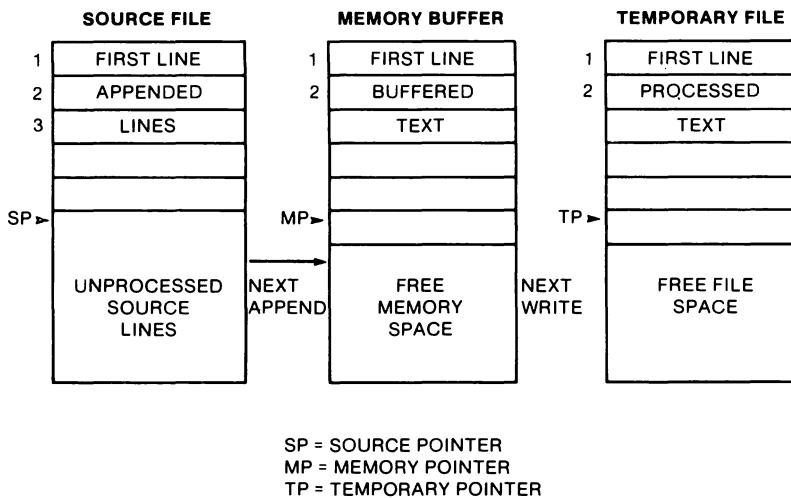


Figure 2-2. Memory Buffer Organization

### 2.1.2 Text Transfer Functions

Given that n is an integer value in the range 0 through 65535, several single-letter ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file. Single letter commands are shown in upper-case, but can be typed in either upper- or lower-case.

Table 2-1. ED Text Transfer Commands

Command	Result
nA	Appends the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n. If upper-case translation is set (see the U command) and the A command is typed in upper-case, all input lines will automatically be translated to upper-case.
nW	Writes the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n + 1 through MP to the top of the memory buffer. Increment TP by n.

Table 2-1. (continued)

Command	Result
E	Ends the edit. Copy all buffered text to temporary file and copy all unprocessed source lines to temporary file. Rename files.
H	Moves to head of new file by performing automatic E command. The temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created. The effect is equivalent to issuing an E command, followed by a reinvocation of ED, using x.y as the file to edit.
O	Returns to original file. The memory buffer is emptied, the temporary file is deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.
Q	Quits edit with no file alterations, returns to CP/M.

There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write one line, respectively. In addition, if a pound sign # is given in the place of n, then the integer 65535 is assumed (the largest value for n that is allowed). Because most source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file.

Two special forms of the A and W commands are provided as a convenience. The command 0A fills the current memory buffer at least half full, while 0W writes lines until the buffer is at least half empty. An error is issued if the memory buffer size is exceeded. You can then enter any command, such as W, that does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

### 2.1.3 Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an imaginary character pointer (CP) that moves throughout the memory buffer under command of the operator.

The memory buffer appears logically as shown in Figure 2-3, where the dashes represent characters of the source line of indefinite length, terminated by carriage return (<cr>) and line-feed (<lf>) characters, and CP represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line that contains the CP.

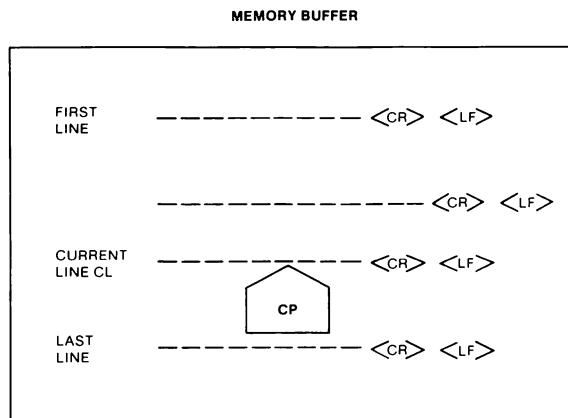


Figure 2-3. Logical Organization of Memory Buffer

#### 2.1.4 Line Numbers and ED Start-up

ED produces absolute line number prefixes that are used to reference a line or range of lines. The absolute line number is displayed at the beginning of each line when ED is in insert mode (see the I command in Section 2.1.5). Each line number takes the form

nnnnn:

where nnnnn is an absolute line number in the range of 1 to 65535. If the memory buffer is empty or if the current line is at the end of the memory buffer, nnnnn appears as 5 blanks.

You can reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. The line denoted by the absolute line number must be in the memory buffer (see the A command). Thus, the command

**345:T**

is interpreted as move to absolute 345, and type the line. Absolute line numbers are produced only during the editing process and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

You can also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute number by a colon. Thus, the command

**:400T**

is interpreted as type from the current line number through the line whose absolute number is 400. Combining the two line reference forms, the command

**345::400T**

is interpreted as move to absolute line 345, then type through absolute line 400. Absolute line references of this sort can precede any of the standard ED commands.

Line numbering is controlled by the V (Verify Line Numbers) command. Line numbering can be turned off by typing the -V command.

If the file to edit does not exist, ED displays the following message:

**NEW FILE**

To move text into the memory buffer, you must enter an i command before typing input lines and terminate each line with a carriage return. A single CTRL-Z character returns ED to command mode.

### 2.1.5 Memory Buffer Operation

When ED begins, the memory buffer is empty. You can either append lines from the source file with the A command, or enter the lines directly from the console with the insert command. The insert command takes the following form:

I

ED then accepts any number of input lines. You must terminate each line with a <cr> (the <lf> is supplied automatically). A single CTRL-Z, denoted by an up arrow ( $\uparrow$ )Z, returns ED to command mode. The CP is positioned after the last character entered. The following sequence:

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
 $\uparrow$ Z
```

leaves the memory buffer as

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```

Generally, ED accepts command letters in upper- or lower-case. If the command is upper-case, all input values associated with the command are translated to upper-case. If the I command is typed, all input lines are automatically translated internally to upper-case. The lower-case form of the i command is most often used to allow both upper- and lower-case letters to be entered.

Various commands can be issued that control the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an optional unsigned value can be specified. When preceded by  $\pm$ , the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign # is replaced by 65535. If an integer n is optional, but not supplied, then n=1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

Table 2-2. Editing Commands

<i>Command</i>	<i>Action</i>
$\pm B$	Move CP to beginning of memory buffer if + and to bottom if -.
$\pm nC$	Move CP by $\pm n$ characters (moving ahead if +), counting the <cr><lf> as two characters.
$\pm nD$	Delete n characters ahead of CP if plus and behind CP if minus.
$\pm nK$	Kill (remove) $\pm n$ lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, the characters before CP remain if + is specified, while the characters after CP remain if - is given in the command.
$\pm nL$	If $n = 0$ , move CP to the beginning of the current line, if it is not already there. If $n \neq 0$ , first move the CP to the beginning of the current line and then move it to the beginning of the line that is n lines down (if +) or up (if -). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.
$\pm nT$	If $n = 0$ , type the contents of the current line up to CP. If $n = 1$ , type the contents of the current line from CP to the end of the line. If $n > 1$ , type the current line along with $n - 1$ lines that follow, if + is specified. Similarly, if $n > 1$ and - is given, type the previous n lines up to the CP. Any key can be depressed to abort long type-outs.
$\pm n$	Equivalent to $\pm nLT$ , which moves up or down and types a single line.

### 2.1.6 Command Strings

Any number of commands can be typed contiguously (up to the capacity of the console buffer) and are executed only after you press the <cr>. Table 2-3 summarizes the CP/M console line-editing commands used to control the input command line.

Table 2-3. Line-editing Controls

<i>Command</i>	<i>Result</i>
CTRL-C	Reboots the CP/M system when typed at the start of a line.
CTRL-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
CTRL-H	Backspaces one character position.
CTRL-J	Terminates current input (line-feed).
CTRL-M	Terminates current input (carriage return).
CTRL-R	Retypes current command line: types a clean line character deletion with rubouts.
CTRL-U	Deletes the entire line typed at the console.
CTRL-X	Same as CTRL-U.
CTRL-Z	Ends input from the console (used in PIP and ED).
rub/del	Deletes and echos the last character typed at the console.

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. In the following example, the command strings on the left produce the results shown to the right. Use lower-case command letters to avoid automatic translation of strings to upper-case.

*Command String**Effect***B2T<cr>**

Move to beginning of the buffer and type two lines:

```
NOW IS THE
TIME FOR
```

The result in the memory buffer is

```
^NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```

**5C0T<cr>**

Move CP five characters and type the beginning of the line NOW I. The result in the memory buffer is

```
NOW I`S THE<cr><lf>
```

**2L-T<cr>**

Move two lines down and type the previous line TIME FOR. The result in the memory buffer is

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
^ALL GOOD MEN<cr><lf>
```

**-L#K<cr>**

Move up one line, delete 65535 lines that follow. The result in the memory buffer is

```
NOW IS THE<cr><lf>^
```

```
I<cr>
TIME TO<cr>
INSERT<cr>
†z
```

Insert two lines of text with automatic translation to upper-case. The result in the memory buffer is

```
NOW IS THE<cr><lf>
TIME TO<cr><lf>
^INSERT<cr><lf>^
```

<i>Command String</i>	<i>Effect</i>
-2L*T<cr>	Move up two lines and type 65535 lines ahead of CP NOW IS THE. The result in the memory buffer is  NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>
<cr>	Move down one line and type one line INSERT. The result in the memory buffer is  NOW IS THE<cr><lf> TIME TO<cr><lf> ^INSERT<cr><lf>

### 2.1.7 Text Search and Alteration

ED has a command that locates strings within the memory buffer. The command takes the form

nF s <cr>

or

nF s ↑Z

where s represents the string to match, followed by either a <cr> or CTRL-Z, denoted by ↑Z. ED starts at the current position of CP and attempts to match the string. The match is attempted n times and, if successful, the CP is moved directly after the string. If the n matches are not successful, the CP is not moved from its initial position. Search strings can include CTRL-L, which is replaced by the pair of symbols <cr><lf>.

The following commands illustrate the use of the F command:

<i>Command String</i>	<i>Effect</i>
B*T<cr>	Move to the beginning and type the entire buffer. The result in the memory buffer is  ^NOW IS THE <cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
F S T<cr>	Find the end of the string S T. The result in the memory buffer is  NOW IS T^HE<cr><lf>
F I s↑ZOTT	Find the next I and type to the CP; then type the remainder of the current line ME FOR. The result in the memory buffer is  NOW IS THE<cr><lf> TI^ME FOR<cr><lf>  ALL GOOD MEN<cr><lf>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is

Is↑Z

or

Is<cr>

where s is the string to insert. If the insertion string is terminated by a CTRL-Z, the string is inserted directly following the CP, and the CP is positioned directly after the string. The action is the same if the command is followed by a <cr> except that a <cr><lf> is automatically inserted into the text following the string. The following command sequences are examples of the F and I commands:

*Command String**Effect*

BIT HIS IS ↑Z &lt;cr&gt;

Insert THIS IS at the beginning of the text. The result in the memory buffer is

THIS IS ^ NOW THE&lt;cr&gt;&lt;lf&gt;

TIME FOR<cr><lf>  
ALL GOOD MEN<cr><lf>

FTIME↑Z-4DIPPLACE↑Z &lt;cr&gt;

Find TIME and delete it; then insert PLACE. The result in the memory buffer is

THIS IS NOW THE<cr><lf>  
PLACE ^ FOR<cr><lf>

ALL GOOD MEN&lt;cr&gt;&lt;lf&gt;

3FO↑Z-3D5D1  
CHANGES↑Z <cr>

Find third occurrence of O (that is, the second O in GOOD), delete previous 3 characters and the subsequent 5 characters; then insert CHANGES. The result in the memory buffer is

THIS IS NOW THE<cr><lf>  
PLACE FOR<cr><lf>  
ALL CHANGES ^ <cr><lf>

-8CISOURCE &lt;cr&gt;

Move back 8 characters and insert the line SOURCE&lt;cr&gt;&lt;lf&gt;. The result in the memory buffer is

THIS IS NOW THE<cr><lf>  
PLACE FOR<cr><lf>  
ALL SOURCE<cr><lf>  
^CHANGES<cr><lf>

ED also provides a single command that combines the F and I commands to perform simple string substitutions. The command takes the following form:

nS  $s_1 \uparrow Z s_2 <cr>$

or

nS  $s_1 \uparrow Z s_2 \uparrow Z$

and has exactly the same effect as applying the following command string a total of n times:

F  $s_1 \uparrow Z - kDIs_2 <cr>$

or

.F  $s_1 \uparrow Z - kDIs_2 \uparrow Z$

where k is the length of the string. ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED that automatically appends and writes lines as the search proceeds. The form is

n N s <cr>

or

n N s  $\uparrow Z$

which searches the entire source file for the nth occurrence of the strings (you should recall that F fails if the string cannot be found in the current buffer). The operation of the N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory content is written (that is, an automatic #W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command, takes the form

$n J s_1 \uparrow Z s_2 \uparrow Z s_3 <cr>$

or

$n J s_1 \uparrow Z s_2 \uparrow Z s_3 \uparrow Z$

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string  $s_1$ . If found, insert the string  $s_2$ , and move CP to follow  $s_2$ . Then delete all characters following CP up to, but not including, the string  $s_3$ , leaving CP directly after  $s_2$ . If  $s_3$  cannot be found, then no deletion is made. If the current line is

NOW IS THE TIME <cr><lf>

the command

JW $\uparrow$ ZWHAT $\uparrow$ Z $\uparrow$ 1 <cr>

results in

NOW WHAT <cr><lf>

You should recall that  $\uparrow$ l (CTRL-L) represents the pair <cr><lf> in search and substitute strings.

The number of characters ED allows in the F, S, N, and J commands is limited to 100 symbols.

### 2.1.8 Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

R filename  $\uparrow$ Z

or

R filename <cr>

where filename is the primary filename of a source file on the disk with an assumed filetype of LIB. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

RMACRO <cr>

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file and automatically inserts the characters into the memory buffer.

ED also includes a block move facility implemented through the X (Transfer) command. The form

nX

transfers the next n lines from the current line to a temporary file called

X\$\$\$\$\$\$,LIB

which is active only during the editing process. You can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred lines accumulate one after another in this file and can be retrieved by simply typing

R

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred LIB file. That is, given that a set of lines has been transferred with the X command, they can be reread any number of times back into the source file. The command

OX

is provided to empty the transferred line file.

Note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted with a CTRL-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

### 2.1.9 Repetitive Command Execution

The macro command M allows you to group ED commands together for repeated evaluation. The M command takes the following form:

n M CS <cr>

or

n M CS ↑Z

where CS represents a string of ED commands, not including another M command. ED executes the command string n times if  $n > 1$ . If  $n = 0$  or 1, the command string is executed repetitively until an error condition is encountered (for example, the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line that is changed:

MFGAMMAT↑Z-5D IDELTA↑ZOTT <cr>

or equivalently

MSGAMMAT↑ZDELTA↑ZOTT <cr>

## 2.2 ED Error Conditions

On error conditions, ED prints the message **BREAK X AT C** where X is one of the error indicators shown in Table 2-4.

**Table 2-4. Error Message Symbols**

Symbol	Meaning
?	Unrecognized command.
>	Memory buffer full (use one of the commands D, K, N, S, or W to remove characters); F, N, or S strings too long.
#	Cannot apply command the number of times specified (for example, in F command).
O	Cannot open LIB file in R command.

If there is a disk error, CP/M displays the following message:

**BDOS ERR on d: BAD SECTOR**

You can choose to ignore the error by pressing RETURN at the console (in this case, the memory buffer data should be examined to see if they were incorrectly read), or you can reset the system with a CTRL-C and reclaim the back-up file if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information. For example, type the following:

**TYPE x.BAK**

where x is the file being edited. Then remove the primary file

**ERA x.y**

and rename the BAK file

**REN x.y = x.BAK**

The file can then be reedited, starting with the previous version.

ED also takes file attributes into account. If you attempt to edit a Read-Only file, the message

**\*\* FILE IS READ/ONLY \*\***

appears at the console. The file can be loaded and examined, but cannot be altered. You must end the edit session and use STAT to change the file attribute to R/W. If the edited file has the system attribute set, the following message:

**'SYSTEM' FILE NOT ACCESSIBLE**

is displayed and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

## 2.3 Control Characters and Commands

Table 2-5 summarizes the control characters and commands available in ED.

Table 2-5. ED Control Characters

<i>Control Character</i>	<i>Function</i>
CTRL-C	System reboot
CTRL-E	Physical <cr><lf> (not actually entered in command)
CTRL-H	Backspace
CTRL-J	Logical tab (cols 1, 9, 16, . . .)
CTRL-L	Logical <cr><lf> in search and substitute strings
CTRL-R	Repeat line
CTRL-U	Line delete
CTRL-X	Line delete
CTRL-Z	String terminator
rub/del	Character delete

Table 2-6 summarizes the commands used in ED.

Table 2-6. ED Commands

<i>Command</i>	<i>Function</i>
nA	Append lines
$\pm$ B	Begin or bottom of buffer
$\pm$ nC	Move character positions
$\pm$ nD	Delete characters
E	End edit and close files (normal end)
nF	Find string
H	End edit, close and reopen files
I	Insert characters, use i if both upper- and lower-case characters are to be entered.
nJ	Place strings in juxtaposition
$\pm$ nK	Kill lines
$\pm$ nL	Move down/up lines
nM	Macro definition
nN	Find next occurrence with autoscan
O	Return to original file
$\pm$ nP	Move and print pages
Q	Quit with no file changes
R	Read library file

Table 2-6. (continued)

<i>Command</i>	<i>Function</i>
nS	Substitute strings
$\pm$ nT	Type lines
$\pm$ U	Translate lower- to upper-case if U, no translation if –U
$\pm$ V	Verify line numbers, or show remaining free character space
0V	A special case of the V command, 0V, prints the memory buffer statistics in the form free/total  where free is the number of free bytes in the memory buffer (in decimal) and total is the size of the memory buffer
nW	Write lines
nZ	Wait (sleep) for approximately n seconds
$\pm$ n	Move and type ( $\pm$ nLT).

Because of common typographical errors, ED requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The following commands:

- E(end)
- H(head)
- O(original)
- Q(quit)

must be typed as single letter commands.

The commands I, J, M, N, R, and S should be typed as i, j, m, n, r, and s if both upper- and lower-case characters are used in the operation, otherwise all characters are converted to upper-case. When a command is entered in upper-case, ED automatically converts the associated string to upper-case, and vice versa.

*End of Section 2*

# Section 3

## CP/M Assembler

### 3.1 Introduction

The CP/M assembler reads assembly-language source files from the disk and produces 8080 machine language in Intel hex format. To start the CP/M assembler, type a command in one of the following forms:

`ASM filename`  
`ASM filename.parms`

In both cases, the assembler assumes there is a file on the disk with the name:

`filename.ASM`

which contains an 8080 assembly-language source file. The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access and hex and print file destinations.

In either case, the CP/M assembler loads and prints the message:

`CP/M ASSEMBLER VER n.n`

where `n.n` is the current version number. In the case of the first command, the assembler reads the source file with assumed filetype ASM and creates two output files

`filename.HEX`  
`filename.PRN`

The HEX file contains the machine code corresponding to the original program in Intel hex format, and the PRN file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they are listed in the PRN file and at the console.

The form ASM filename parms is used to redirect input and output files from their defaults. In this case, the parms portion of the command is a three-letter group that specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

filename.p1p2p3

where p1, p2, and p3 are single letters. P1 can be

A,B,...,P

which designates the disk name that contains the source file. P2 can be

A,B,...,P

which designates the disk name that will receive the hex file; or, P2 can be

Z

which skips the generation of the hex file.

P3 can be

A,B,...,P

which designates the disk name that will receive the print file. P3 can also be specified as

X

which places the listing at the console; or

Z

which skips generation of the print file. Thus, the command

ASM X,AAA

indicates that the source, X.HEX, and print, X.PRN files are also to be created on disk A. This form of the command is implied if the assembler is run from disk A. Given that you are currently addressing disk A, the above command is the same as

ASM X

### The command

**ASM X.ABX**

indicates that the source file is to be taken from disk A, the hex file is to be placed on disk B, and the listing file is to be sent to the console. The command

**ASM X.BZZ**

takes the source file from disk B and skips the generation of the hex and print files. This command is useful for fast execution of the assembler to check program syntax.

The source program format is compatible with the Intel 8080 assembler. Macros are not implemented in ASM; see the optional MAC macro assembler. There are certain extensions in the CP/M assembler that make it somewhat easier to use. These extensions are described below.

## 3.2 Program Format

An assembly-language program acceptable as input to the assembler consists of a sequence of statements of the form

line# label operation operand ;comment

where any or all of the fields may be present in a particular instance. Each assembly-language statement is terminated with a carriage return and line-feed (the line-feed is inserted automatically by the ED program), or with the character !, which is treated as an end-of-line by the assembler. Thus, multiple assembly-language statements can be written on the same physical line if separated by exclamation point symbols.

The line# is an optional decimal integer value representing the source program line number, and ASM ignores this field if present.

The label field takes either of the following forms:

identifier  
identifier:

The label field is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters where the first character is alphabetic. Identifiers can be freely used by the programmer to label elements such as

program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol \$, which can be used to improve readability of the name. Further, all lower-case alphabetics are treated as upper-case. The following are all valid instances of labels:

x	xy	long\$name
x:	yx1:	longer\$named\$data:
X1Y2	X1x2	x234\$5678\$9012\$3456:

The operation field contains either an assembler directive or pseudo operation, or an 8080 machine operation code. The pseudo operations and machine operation codes are described in Section 3.3.

Generally, the operand field of the statement contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. Again, the complete details of properly formed expressions are given in Section 3.3.

The comment field contains arbitrary characters following the semicolon symbol until the next real or logical end-of-line. These characters are read, listed, and otherwise ignored by the assembler. The CP/M assembler also treats statements that begin with an \* in column one as comment statements that are listed and ignored in the assembly process.

The assembly-language program is formulated as a sequence of statements of the above form, terminated by an optional END statement. All statements following the END are ignored by the assembler.

### 3.3 Forming the Operand

To describe the operation codes and pseudo operations completely, it is necessary first to present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands, labels, constants, and reserved words, combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. If an expression is to be used in a byte move immediate instruction, the most significant 8 bits of the expression must be zero. The restriction on the expression significance is given with the individual instructions.

### 3.3.1 Labels

A label is an identifier that occurs on a particular statement. In general, the label is given a value determined by the type of statement that it precedes. If the label occurs on a statement that generates machine code or reserves memory space (for example, a MOV instruction or a DS pseudo operation), the label is given the value of the program address that it labels. If the label precedes an EQU or SET, the label is given the value that results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

### 3.3.2 Numeric Constants

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The following are radix indicators:

- B is a binary constant (base 2).
- O is a octal constant (base 8).
- Q is a octal constant (base 8).
- D is a decimal constant (base 10).
- H is a hexadecimal constant (base 16).

Q is an alternate radix indicator for octal numbers because the letter O is easily confused with the digit 0. Any numeric constant that does not terminate with a radix indicator is a decimal constant.

A constant is composed as a sequence of digits, followed by an optional radix indicator, where the digits are in the appropriate range for the radix. Binary constants must be composed of 0 and 1 digits, octal constants can contain digits in the range 0–7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A(10D), B(11D), C(12D), D(13D), E(14D), and F(15D). Note that the leading digit of a hexadecimal constant must be a decimal digit to avoid confusing a hexadecimal constant with an identifier. A leading 0 will always suffice. A constant composed in this manner must evaluate to a binary number that can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler.

Similar to identifiers, embedded \$ signs are allowed within constants to improve their readability. Finally, the radix indicator is translated to upper-case if a lower-case letter is encountered. The following are all valid instances of numeric constants:

1234	1234D	1100B	1111\$0000\$1111\$0000B
1234H	OFFEH	33770	33\$77\$22Q
33770	0fe3h	1234d	0ffffh

### 3.3.3 Reserved Words

There are several reserved character sequences that have predefined meanings in the operand field of a statement. The names of 8080 registers are given below. When they are encountered, they produce the values shown to the right.

Table 3-1. Reserved Characters

<i>Character</i>	<i>Value</i>
A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

Again, lower-case names have the same values as their upper-case equivalents. Machine instructions can also be used in the operand field; they evaluate to their internal codes. In the case of instructions that require operands, where the specific operand becomes a part of the binary bit pattern of the instruction, for example, MOV A,B, the value of the instruction, in this case MOV, is the bit pattern of the instruction with zeros in the optional fields, for example, MOV produces 40H.

When the symbol \$ occurs in the operand field, not embedded within identifiers and numeric constants, its value becomes the address of the next instruction to generate, not including the instruction contained within the current logical line.

### 3.3.4 String Constants

String constants represent sequences of ASCII characters and are represented by enclosing the characters within apostrophe symbols. All strings must be fully contained within the current physical line (thus allowing exclamation point symbols within strings) and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe (the two keystrokes"), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8- or 16-bit value, respectively. Two-character strings become a 16-bit constant, with the second character as the low-order byte, and the first character as the high-order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings; both upper- and lower-case characters can be represented. You should note that only graphic printing ASCII characters are allowed within strings.

Valid strings:

```
'A' 'AB' 'ab' 'c'  
' ' 'a' ' ' ' ' '  
'Walla Walla Wash.'  
'She said "Hello" to me.'  
'I said "Hello" to her.'
```

How assembler reads strings:

```
A AB ab c  
a '' ''  
Walla Walla Wash.  
She said "Hello" to me.  
I said "Hello" to her.
```

### 3.3.5 Arithmetic and Logical Operators

The operands described in Section 3.3 can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are described in Table 3-2.

Table 3-2. Arithmetic and Logical Operators

Operators	Meaning
a + b	unsigned arithmetic sum of a and b
a - b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
- b	unary minus (identical to 0 - b)

Table 3-2. (continued)

Operators	Meaning
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b.
NOT b	logical inverse of b (all 0s become 1s, 1s become 0s), where b is considered a 16-bit value
a AND b	bit-by-bit logical and of a and b
a OR b	bit-by-bit logical or of a and b
a XOR b	bit-by-bit logical exclusive or of a and b
a SHL b	the value that results from shifting a to the left by an amount b, with zero fill
a SHR b	the value that results from shifting a to the right by an amount b, with zero fill

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one- or two-character strings) or fully enclosed parenthesized subexpressions, like those shown in the following examples:

10+20 10h+37Q L1/3 (L2+4) SHR3

('a' and 5fh)+'0'('B'+B)OR(PSW+M)

(1+(2+c))shr(A-(B+1))

Note that all computations are performed at assembly time as 16-bit unsigned operations. Thus,  $-1$  is computed as  $0 - 1$ , which results in the value 0ffffh (that is, all 1s). The resulting expression must fit the operation code in which it is used. For example, if the expression is used in an ADI (add immediate) instruction, the high-order 8 bits of the expression must be zero. As a result, the operation ADI-1 produces an error

message ( $-1$  becomes `0ffffh`, which cannot be represented as an 8-bit value), while `ADI(-1) AND 0FFH` is accepted by the assembler because the `AND` operation zeros the high-order bits of the expression.

### 3.3.6 Precedence of Operators

As a convenience to the programmer, ASM assumes that operators have a relative precedence of application that allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses that are defined by the relative precedence. The order of application of operators in unparenthesized expressions is listed below. Operators listed first have highest precedence (they are applied first in an unparenthesized expression), while operators listed last have lowest precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression.

\* /MOD SHL SHR

- 1 -

NOTE

AND

OR XOR

Thus, the expressions shown to the left below are interpreted by the assembler as the fully parenthesized expressions shown to the right.

$$a * b + c$$

$$(a^*b) + c$$

a+b\*c

$$a + (b^*c)$$

a MOD b\*c SHL d

((a MOD b)<sup>\*</sup>c) SHL d

a OR b AND NOT c\&d SHL e

a OR (b AND (NOT (c + (d SHL e))))

Balanced, parenthesized subexpressions can always be used to override the assumed parentheses; thus, the last expression above could be rewritten to force application of operators in a different order, as shown:

( a OR b ) AND ( NOT c ) + d SHL e

This results in these assumed parentheses:

( a OR b ) AND ( ( NOT c ) + ( d SHL e ) )

An unparenthesized expression is well-formed only if the expression that results from inserting the assumed parentheses is well-formed.

## 3.4 Assembler Directives

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a pseudo operation that appears in the operation field of the line. The acceptable pseudo operations are shown in Table 3-3.

Table 3-3. Assembler Directives

Directive	Meaning
ORG	set the program or data origin
END	end program, optional start address
EQU	numeric equate
SET	numeric set
IF	begin conditional assembly
ENDIF	end of conditional assembly
DB	define data bytes
DW	define data words
DS	define data storage area

### 3.4.1 The ORG Directive

The ORG statement takes the form:

label ORG expression

- where label is an optional program identifier and expression is a 16-bit expression, consisting of operands that are defined before the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that the programmer is not defining overlapping memory areas. Note that most programs written for the CP/M system begin with an ORG statement of the form:

ORG 100H

- which causes machine code generation to begin at the base of the CP/M transient program area. If a label is specified in the ORG statement, the label is given the value of the expression. This label can then be used in the operand field of other statements to represent this expression.

### 3.4.2 The END Directive

- The END statement is optional in an assembly-language program, but if it is present it must be the last statement. All subsequent statements are ignored in the assembly. The END statement takes the following two forms:

label END

label END expression

- where the label is again optional. If the first form is used, the assembly process stops, and the default starting address of the program is taken as 0000. Otherwise, the expression is evaluated, and becomes the program starting address. This starting address is included in the last record of the Intel-formatted machine code hex file that results from the assembly. Thus, most CP/M assembly-language programs end with the statement:

END 100H

- resulting in the default starting address of 100H (beginning of the transient program area).

### 3.4.3 The EQU Directive

The EQU (equate) statement is used to set up synonyms for particular numeric values. The EQU statement takes the form:

label EQU expression

where the label must be present and must not label any other statement. The assembler evaluates the expression and assigns this value to the identifier given in the label field. The identifier is usually a name that describes the value in a more human-oriented manner. Further, this name is used throughout the program to place parameters on certain functions. Suppose data received from a teletype appears on a particular input port, and data is sent to the teletype through the next output port in sequence. For example, you can use this series of equate statements to define these ports for a particular hardware environment:

TTYBASE	EQU 10H	;BASE PORT NUMBER FOR TTY
TTYIN	EQU TTYBASE	;TTY DATA IN
TTYOUT	EQU TTYBASE+1	;TTY DATA OUT

At a later point in the program, the statements that access the teletype can appear as follows:

IN	TTYIN	;READ TTY DATA TO REG-A
...		
OUT	TTYOUT	;WRITE DATA TO TTY FROM REG-

making the program more readable than if the absolute I/O ports are used. Further, if the hardware environment is redefined to start the teletype communications ports at 7FH instead of 10H, the first statement need only be changed to

TTYBASE	EQU 7FH	;BASE PORT NUMBER FOR TTY
---------	---------	---------------------------

and the program can be reassembled without changing any other statements.

### 3.4.4 The SET Directive

The SET statement is similar to the EQU, taking the form:

label SET expression

except that the label can occur on other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a label with a single value, while the SET statement defines a value that is valid from the current SET statement to the point where the label occurs on the next SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

### 3.4.5 The IF and ENDIF Directives

The IF and ENDIF statements define a range of assembly-language statements that are to be included or excluded during the assembly process. These statements take on the form:

IF expression

statement#1

statement#2

...

statement#n

ENDIF

When encountering the IF statement, the assembler evaluates the expression following the IF. All operands in the expression must be defined ahead of the IF statement. If the expression evaluates to a nonzero value, then statement#1 through statement#n are assembled. If the expression evaluates to zero, the statements are listed but not assembled. Conditional assembly is often used to write a single generic program that includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments, for example, might be part of a program that communicates with either a teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins.

```

TRUE    EQU      0FFFFH   ;DEFINE VALUE OF TRUE
FALSE   EQU      NOT TRUE ;DEFINE VALUE OF FALSE
;
TTY     EQU      TRUE    ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE EQU      10H    ;BASE OF TTY I/O PORTS
CRTBASE EQU      20H    ;BASE OF CRT I/O PORTS
IF      TTY    ;ASSEMBLE RELATIVE TO
;                  ;TTYBASE
CONIN   EQU      TTYBASE ;CONSOLE INPUT
CONOUT  EQU      TTYBASE+1 ;CONSOLE OUTPUT
ENDIF

;      IF      NOT TTY  ;ASSEMBLE RELATIVE TO
;                  ;CRTBASE
CONIN   EQU      CRTBASE ;CONSOLE INPUT
CONOUT  EQU      CRTBASE+1 ;CONSOLE OUTPUT
ENDIF

...
IN      CONIN   ;READ CONSOLE DATA
...
OUT    CONOUT  ;WRITE CONSOLE DATA

```

In this case, the program assembles for an environment where a teletype is connected, based at port 10H. The statement defining TTY can be changed to

```
TTY EQU FALSE
```

and, in this case, the program assembles for a CRT based at port 20H.

### 3.4.6 The DB Directive

The DB directive allows the programmer to define initialized storage areas in single-precision byte format. The DB statement takes the form:

label DB e#1, e#2, ..., e#n

where e#1 through e#n are either expressions that evaluate to 8-bit values (the high-order bit must be zero) or are ASCII strings of length no greater than 64 characters. There is no practical restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions.

**Note:** ASCII characters are always placed in memory with the parity bit reset (0). Also, there is no translation from lower- to upper-case within strings. The optional label can be used to reference the data area throughout the remainder of the program. The following are examples of valid DB statements:

data:       DB       0,1,2,3,4,5  
              DB       data and Offh,5,377Q,1+2+3+4

sign-on:     DB       'please type your name',cr,lf,0  
              DB       'AB'SHR B,'C','DE', AND 7FH

### 3.4.7 The DW Directive

The DW statement is similar to the DB statement except double-precision two-byte words of storage are initialized. The DW statement takes the form:

label       DW       e#1, e#2, ..., e#n

where e#1 through e#n are expressions that evaluate to 16-bit results. Note that ASCII strings of one or two characters are allowed, but strings longer than two characters are disallowed. In all cases, the data storage is consistent with the 8080 processor; the least significant byte of the expression is stored first in memory, followed by the most significant byte. The following are examples of DW statements:

doub:       DW       Offefh,doub+4,signon-\$,255+255  
              DW       'a',5,'ab','CD',6 shl 8 or 11b.

### 3.4.8 The DS Directive

The DS statement is used to reserve an area of uninitialized memory, and takes the form:

label            DS            expression

where the label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the following statement:

```
label:        EQU $   ;LABEL VALUE IS CURRENT CODE LOCATION  
              ORG $+expression   ;MOVE PAST RESERVED AREA
```

## 3.5 Operation Codes

Assembly-language operation codes form the principal part of assembly-language programs and form the operation field of the instruction. In general, ASM accepts all the standard mnemonics for the Intel 8080 microcomputer, which are given in detail in the *Intel 8080 Assembly Language Programming Manual*. Labels are optional on each input line. The individual operators are listed briefly in the following sections for completeness, although the Intel manuals should be referenced for exact operator details. In Tables 3-4 through 3-8, bit values have the following meaning:

- e3 represents a 3-bit value in the range 0–7 that can be one of the predefined registers A, B, C, D, E, H, L, M, SP, or PSW.
- e8 represents an 8-bit value in the range 0–255.
- e16 represents a 16-bit value in the range 0–65535.

These expressions can be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases are noted as they are encountered.

In the sections that follow, each operation code is listed in its most general form, along with a specific example, a short explanation, and special restrictions.

### 3.5.1 Jumps, Calls, and Returns

The Jump, Call, and Return instructions allow several different forms that test the condition flags set in the 8080 microcomputer CPU. The forms are shown in Table 3-4.

Table 3-4. Jumps, Calls, and Returns

<i>Form</i>	<i>Bit Value</i>	<i>Example</i>	<i>Meaning</i>
JMP	e16	JMP L1	Jump unconditionally to label
JNZ	e16	JNZ L2	Jump on nonzero condition to label
JZ	e16	JZ 100H	Jump on zero condition to label
JNC	e16	JNC L1+4	Jump no carry to label
JC	e16	JC L3	Jump on carry to label
JPO	e16	JPO \$+8	Jump on parity odd to label
JPE	e16	JPE L4	Jump on even parity to label
JP	e16	JP GAMMA	Jump on positive result to label
JM	e16	JM a1	Jump on minus to label
CALL	e16	CALL S1	Call subroutine unconditionally
CNZ	e16	CNZ S2	Call subroutine on nonzero condition
CZ	e16	CZ 100H	Call subroutine on zero condition
CNC	e16	CNC S1+4	Call subroutine if no carry set
CC	e16	CC S3	Call subroutine if carry set
CPO	e16	CPO \$+8	Call subroutine if parity odd

Table 3-4. (continued)

<i>Form</i>	<i>Bit Value</i>	<i>Example</i>	<i>Meaning</i>
CPE	e16	CPE \$4	Call subroutine if parity even
CP	e16	CP GAMMA	Call subroutine if positive result
CM	e16	CM b1\$c2	Call subroutine if minus flag
RST	e3	RST 0	Programmed restart, equivalent to CALL 8*e3, except one byte call
RET			Return from subroutine
RNZ			Return if nonzero flag set
RZ			Return if zero flag set
RNC			Return if no carry
RC			Return if carry flag set
RPO			Return if parity is odd
RPE			Return if parity is even
RP			Return if positive result
RM			Return if minus flag is set

### 3.5.2 Immediate Operand Instructions

Several instructions are available that load single- or double-precision registers or single-precision memory cells with constant values, along with instructions that perform immediate arithmetic or logical operations on the accumulator (register A). Table 3-5 describes the immediate operand instructions.

Table 3-5. Immediate Operand Instructions

<i>Form with Bit Values</i>	<i>Example</i>	<i>Meaning</i>
MVI e3,e8	MVI B ,255	Move immediate data to register A, B, C, D, E, H, L, or M (memory)
ADI e8	ADI 1	Add immediate operand to A without carry
ACI e8	ACI OFFH	Add immediate operand to A with carry
SUI e8	SUI L + 3	Subtract from A without borrow (carry)
SBI e8	SBI L AND 11B	Subtract from A with borrow (carry)
ANI e8	ANI \$ AND 7FH	Logical and A with immediate data
XRI e8	XRI 1111\$0000B	Exclusive or A with immediate data
ORI e8	ORI L AND 1+1	Logical or A with immediate data
CPI e8	CPI 'a'	Compare A with immediate data, same as SUI except register A not changed.
LXI e3,e16	LXI B ,100H	Load extended immediate to register pair. e3 must be equivalent to B, D, H, or SP.

### 3.5.3 Increment and Decrement Instructions

The 8080 provides instructions for incrementing or decrementing single- and double-precision registers. The instructions are described in Table 3-6.

**Table 3-6. Increment and Decrement Instructions**

<i>Form with Bit Value</i>	<i>Example</i>	<i>Meaning</i>
INR e3	INR E	Single-precision increment register. e3 produces one of A, B, C, D, E, H, L, M.
DCR e3	DCR A	Single-precision decrement register. e3 produces one of A, B, C, D, E, H, L, M.
INX e3	INX SP	Double-precision increment register pair. e3 must be equivalent to B, D, H, or SP.
DCX e3	DCX B	Double-precision decrement register pair. e3 must be equivalent to B, D, H, or SP.

### 3.5.4 Data Movement Instructions

Instructions that move data from memory to the CPU and from CPU to memory are given in the following table.

Table 3-7. Data Movement Instructions

<i>Form with Bit Value</i>	<i>Example</i>	<i>Meaning</i>
MOV e3,e3	MOV A,B	Move data to leftmost element from rightmost element. e3 produces one of A, B, C, D, E, H, L, or M. MOV M,M is disallowed.
LDAX e3	LDAX B	Load register A from computed address. e3 must produce either B or D.
STAX e3	STAX D	Store register A to computed address. e3 must produce either B or D.
LHLD e16	LHLD L1	Load HL direct from location e16. Double-precision load to H and L.
SHLD e16	SHLD L5+x	Store HL direct to location e16. Double-precision store from H and L to memory.
LDA e16	LDA Gamma	Load register A from address e16.
STA e16	STA X3-5	Store register A into memory at e16.
POP e3	POP PSW	Load register pair from stack, set SP. e3 must produce one of B, D, H, or PSW.
PUSH e3	PUSH B	Store register pair into stack, set SP. e3 must produce one of B, D, H, or PSW.

Table 3-7. (continued)

<i>Form with Bit Value</i>	<i>Example</i>	<i>Meaning</i>
IN e8	IN 0	Load register A with data from port e8.
OUT e8	OUT 255	Send data from register A to port e8.
XTHL		Exchange data from top of stack with HL.
PCHL		Fill program counter with data from HL.
SPHL		Fill stack pointer with data from HL.
XCHG		Exchange DE pair with HL pair.

### 3.5.5 Arithmetic Logic Unit Operations

Instructions that act upon the single-precision accumulator to perform arithmetic and logic operations are given in the following table.

Table 3-8. Arithmetic Logic Unit Operations

<i>Form with Bit Value</i>	<i>Example</i>	<i>Meaning</i>
ADD e3	ADD B	Add register given by e3 to accumulator without carry. e3 must produce one of A, B, C, D, E, H, or L.
ADC e3	ADC L	Add register to A with carry, e3 as above.
SUB e3	SUB H	Subtract reg e3 from A without carry, e3 is defined as above.

Table 3-8. (continued)

<i>Form with Bit Value</i>	<i>Example</i>	<i>Meaning</i>
SBB e3	SBB Z	Subtract register e3 from A with carry, e3 defined as above.
ANA e3	ANA 1+1	Logical and reg with A, e3 as above.
XRA e3	XRA A	Exclusive or with A, e3 as above.
ORA e3	ORA B	Logical or with A, e3 defined as above.
CMP e3	CMP H	Compare register with A, e3 as above.
DAA		Decimal adjust register A based upon last arithmetic logic unit operation.
CMA		Complement the bits in register A.
STC		Set the carry flag to 1.
CMC		Complement the carry flag.
RLC		Rotate bits left, (re)set carry as a side effect. High-order A bit becomes carry.
RRC		Rotate bits right, (re)set carry as side effect. Low-order A bit becomes carry.
RAL		Rotate carry/A register to left. Carry is involved in the rotate.
RAR		Rotate carry/A register to right. Carry is involved in the rotate.
DAD e3	DAD B	Double-precision add register pair e3 to HL. e3 must produce B, D, H, or SP.

### 3.5.6 Control Instructions

The four remaining instructions, categorized as control instructions, are the following:

- HLT halts the 8080 processor.
- DI disables the interrupt system.
- EI enables the interrupt system.
- NOP means no operation.

## 3.6 Error Messages

When errors occur within the assembly-language program, they are listed as single-character flags in the leftmost position of the source listing. The line in error is also echoed at the console so that the source listing need not be examined to determine if errors are present. The error codes are listed in the following table.

Table 3-9. Error Codes

Error Code	Meaning
D	Data error: element in data statement cannot be placed in the specified data area.
E	Expression error: expression is ill-formed and cannot be computed at assembly time.
L	Label error: label cannot appear in this context; might be duplicate label.
N	Not implemented: features that will appear in future ASM versions. For example, macros are recognized, but flagged in this version.
O	Overflow: expression is too complicated (too many pending operators) to be computed and should be simplified.
P	Phase error: label does not have the same value on two subsequent passes through the program.

Table 3-9. (continued)

Error Code	Meaning
R	Register error: the value specified as a register is not compatible with the operation code.
S	Syntax error: statement is not properly formed.
V	Value error: operand encountered in expression is improperly formed.

Table 3-10 lists the error messages that are due to terminal error conditions.

Table 3-10. Error Messages

Message	Meaning
NO SOURCE FILE PRESENT	The file specified in the ASM command does not exist on disk.
NO DIRECTORY SPACE	The disk directory is full; erase files that are not needed and retry.
SOURCE FILE NAME ERROR	Improperly formed ASM filename, for example, it is specified with ? field s.
SOURCE FILE READ ERROR	Source file cannot be read properly by the assembler; execute a TYPE to determine the point of error.
OUTPUT FILE WRITE ERROR	Output files cannot be written properly; most likely cause is a full disk, erase and retry.
CANNOT CLOSE FILE	Output file cannot be closed; check to see if disk is write protected.

### 3.7 A Sample Session

The following sample session shows interaction with the assembler and debugger in the development of a simple assembly-language program. The ↴ arrow represents a carriage return keystroke.

A>ASM SORT ↴ Assemble SORT.ASM

CP/M ASSEMBLER - VER 1.0

0015C Next free address

003H USE FACTOR Percent of table used 00 to ff (hexadecimal)

END OF ASSEMBLY

A>DIR SORT,\* ↴

SORT ASM Source file

SORT BAK Back-up from last edit

SORT PRN Print file (contains tab characters)

SORT HEX Machine code file

A>TYPE SORT,PRN ↴

Source line

; SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE  
 ; START AT THE BEGINNING OF THE TRANSIENT  
 ; PROGRAM AREA

Machine code location

0100 ← ORG 100H

Generated machine code

0100 214601	SORT:	LXI H,SW	; ADDRESS SWITCH TOGGLE
0103 3601		MVI M,1	; SET TO 1 FOR FIRST ITERATION
0105 214701		LXI H,I	; ADDRESS INDEX
0108 3600		MVI M,0	; I=0
;			
; COMPARE I WITH ARRAY SIZE			
010A 7E	COMPL:	MOV A,M	; A REGISTER = I
010B FE09		CPI N-1	; CY SET IF I<(N-1)
010D D21901		JNC CONT	; CONTINUE IF I=(N-2)
;			
; END OF ONE PASS THROUGH DATA			

```

0110 214601      LXI H,SW ;CHECK FOR ZERO SWITCHES
0113 7EB7C200001  MOV A, M! ORA A! JNZ SORT ;END OF SORT IF SW=0
;
0118 FF          RST 7    ;GO TO THE DEBUGGER INSTEAD OF REB
;
;           ;CONTINUE THIS PASS
Truncated ; ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
0119
5F16002148CONT: MOV E, A! MVI D, 0! LXI H, AV! DAD D! DAD D
0121 4E792346  MOV C, M! MOV A, C! INX H! MOV B, M
;           ;LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B
;
;           ;MOV H AND L TO ADDRESS AV(I+1)
0125 23          INX H
;
;           ;COMPARE VALUE WITH REGS CONTAINING AV (I)
0126 965778239E  SUB M! MOV D, A! MOV A, B! INX H! SBB M ;SUBTRACT
;
;           ;BORROW SET IF AV(I+1)>AV(I)
012B DA3F01      JC INCI ;SKIP IF IN PROPER ORDER
;
;           ;CHECK FOR EQUAL VALUES
012E B2CA3F01  ORA D! JZ INCI ;SKIP IF AV(I) = AV(I+1)
0132 56702B5E  MOV D, M! MOV M, B! DCX H! MOV E, M
0136 712B722B73  MOV M, C! DCX H! MOV M, D! DCX H! MOV M, E
;
;           ;INCREMENT SWITCH COUNT
013B 21460134  LXI H,SW! INR M
;
;           ;INCREMENT I
013F 21470134C3INCI:LXI H,I! INR M! JMP COMP
;
;           ;DATA DEFINITION SECTION
0146 00      SW:   DB 0      ;RESERVE SPACE FOR SWITCH COUNT
0147    I:   DS 1      ;SPACE FOR INDEX
0148 050064001EAV: DW 5, 100, 30, 50, 20, 7, 1000, 300, 100, -32767
000A = N      EQU($-AV)/2 ;COMPUTE N INSTEAD OF PRE
015C          END
A>TYPE SORT.HEX  Equate value

```

:10010000214601360121470136007EFE09D2190140  
 :100110002146017EB7C20001FF5F16002148011988 } Machine code in  
 :10012000194E79234623965778239EDA3F01B2CAA7 } HEX format

:100130003F0156702B5E712B722B732146013421C7  
 :07014000470134C30A01006E  
 :10014800050064001E00320014000700E8032C01BB } Machine code in  
 :0401580064000180BE  
 :000000000000 } HEX format

A>DDT SORT.HEX Start debug run

16K DDT VER 1.0

NEXT PC

015C 0000 Default address (no address on END statement)

-XP

P=0000 100 Change PC to 100

-UFFFF Untrace for 65535 steps

Abort with rubout

C0ZOMOE0IO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H, 0146\*0100  
 -T10 Trace 10<sub>16</sub> steps

C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M  
 C0ZOMOE0IO A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09  
 C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=010D JNC 0119  
 C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=0110 LXI H, 0146  
 C1ZOM1E0IO A=00 B=0000 D=0000 H=0146 S=0100 P=0113 MOV A, M  
 C1ZOM1E0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0114 ORA A  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0115 JNZ 0100  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00  
 C0ZOMOE0IO A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M\*010B  
 -A10D Stopped at 10BH

010D JC 119, Change to a jump on carry

0110,

-XP,

P=010B 100, Reset program counter back to beginning of program

-T10, Trace execution for 10H steps

#### Altered instruction

```

COZOMOE0IO A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
COZOMOE0IO A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
COZOMOE0IO A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
COZOMOE0IO A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
COZOMOE0IO A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M
COZOMOE0IO A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=010D JC 0119
C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=0119 MOV E,A
C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=011A MVI D,00
C1ZOM1E0IO A=00 B=0000 D=0000 H=0147 S=0100 P=011C LXI H,0148
C1ZOM1E0IO A=00 B=0000 D=0000 H=0148 S=0100 P=011F DAD D
COZOM1E0IO A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD D
COZOM1E0IO A=00 B=0000 D=0000 H=0148 S=0100 P=0121 MOV C,M
COZOM1E0IO A=00 B=0005 D=0000 H=0148 S=0100 P=0122 MOV A,C
COZOM1E0IO A=05 B=0005 D=0000 H=0148 S=0100 P=0123 INX H
COZOM1E0IO A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOV B,M*0125
-L100, Automatic breakpoint

```

```

0100 LXI H,0146
0103 MVI M,01
0105 LXI H,0147
0108 MVI M,00
010A MOV A,M
010B CPI 09
010D JC 0119
0110 LXI H,0146
0113 MOV A,M
0114 ORA A
0115 JNZ 0100
-L,

```

List some code  
from 100H

```

0118 RST 07
0119 MOV E,A
011A MVI D,00
011C LXI H,0148
    }
List more

```

-Abort list with rubout

-G,11B ↗ Start program from current PC (0125H) and run in real time to 11BH

\*0127 Stopped with an external interrupt 7 from front panel

-T4 ↗ (program was looping indefinitely)

Look at looping program in trace mode

```

COZOMOE0IO A=38 B=0064 D=0006 H=0156 S=0100 P=0127 MOV D,A
COZOMOE0IO A=38 B=0064 D=3806 H=0156 S=0100 P=0128 MOV A,B
COZOMOE0IO A=00 B=0064 D=3806 H=0156 S=0100 P=0129 INX H
COZOMOE0IO A=00 B=0064 D=3806 H=0157 S=0100 P=012A SBB M*012B
-D148 ↗

```

Data are sorted, but program does not stop.

0148 05 00 07 00 14 00 1E 00.....

0150 32 00 B4 00 B4 00 2C 01 E8 03 01 B0 00 00 00 00 2,D,D,.....

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00,.....

-GO ↗ Return to CP/M

A>DDT SORT,HEX ↗ Reload the memory image

16K DDT VER 1.0

NEXT PC

015C 0000

-XP

P=0000 100 ↗ Set PC to beginning of program

-L100 ↗ List bad OPCODE

010D JNC 0115

0110 LXI H,0146

-Abort list with rubout

-A100 ↗ Assemble new OPCODE

010D JC 119✓

0110✓

-L100✓ List starting section of program

0100 LXI H,0146

0103 MVI M,01

0105 LXI H,0147

0108 MVI M,00

-Abort list with rubout

-A103✓ Change switch initialization to 00

0103 MVI M,0✓

0105✓

-^C Return to CP/M with CTRL-C (G0 works as well)

SAVE 1 SORT.COM✓ Save 1 page (256 bytes, from 100H to 1ffH) on disk in case there is need to reload later

A>DDT SORT.COM✓ Restart DDT with saved memory image

16K DDT VER 1.0

NEXT PC

0200 0100 COM file always starts with address 100H

-G✓ Run the program from PC=100H

\*0118 Programmed stop (RST 7) encountered

-D148

Data properly sorted

0148 05 00 07 00 14 00 1E 00.....  
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 00 2,D,D.....

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.....  
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.....

-G0✓Return to CP/M

A>ED SORT.ASM ↗ Make changes to original program  
 (The caret, ^, indicates a control character)

\*N,0^ZOTT ↗ Find next ,0  
 MVI M,0 ;I = 0

\*-↗ Up one line in text  
 LXI H,I ;ADDRESS INDEX

\*-↗ Up another line  
 MVI M,1 ;SET TO 1 FOR FIRST ITERATION

\*KT ↗ Kill line and type next line  
 LXI H,I ;ADDRESS INDEX

\*I ↗ Insert new line  
 MVI M,0 ;ZERO SW

\*T ↗  
 LXI H,I ;ADDRESS INDEX

\*NJNC ^ZOT ↗  
 JNC\*T ↗  
 CONT ↗ ;CONTINUE IF I<=(N-2)

\*-2DIC ^ZOLT ↗  
 JC CONT ;CONTINUE IF I<=(N-2)

\*E ↗  
 └─ Source from disk A  
 └─ HEX to disk A  
 A>ASM SORT.AAZ ↗ Skip PRN file

CP/M ASSEMBLER - VER 1.0

015C Next address to assemble  
 003H USE FACTOR  
 END OF ASSEMBLY

A>DDT SORT.HEX Test program changes

16K DDT VER 1.0

NEXT PC

015C 0000

-G100

\*0118

-D148

→ Data sorted

0148 05 00 07 00 14 00 1E 00.....

0150 32 00 64 00 64 00 2C 01 E8 03 01 B0 00 00 00 00 2,D,D.....

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.....

-Abort with rubout

-GO → Return to CP/M—program checks OK.

*End of Section 3*

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

ל

# Section 4

## CP/M Dynamic Debugging Tool

### 4.1 Introduction

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. Invoke the debugger with a command of one of the following forms:

```
DDT  
DDT filename.HEX  
DDT filename.COM
```

where filename is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in place of the Console Command Processor (CCP) and resides directly below the Basic Disk Operating System (BDOS) portion of CP/M. Refer to Section 5 for standard memory organization. The BDOS starting address, located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area (TPA) size.

The second and third forms of the DDT command perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the following sequence of commands:

```
DDT  
Ifilename.HEX or Ifilename.COM  
R
```

where the I and R commands set up and read the specified program to test. See the explanation of the I and R commands below for exact details.

Upon initiation, DDT prints a sign-on message in the form:

```
DDT VER m.m
```

where m.m is the revision number.

Following the sign-on message, DDT prompts you with the hyphen character, -, and waits for input commands from the console. You can type any of several single-character commands, followed by a carriage return to execute the command. Each line of input can be line-edited using the following standard CP/M controls:

**Table 4-1. Line-editing Controls**

<i>Control</i>	<i>Result</i>
rubout	removes the last character typed
CTRL-U	removes the entire line, ready for retyping
CTRL-C	reboots system

Any command can be up to 32 characters in length. An automatic carriage return is inserted as character 33, where the first character determines the command type. Table 4-2 describes DDT commands.

**Table 4-2. DDT Commands**

<i>Command Character</i>	<i>Result</i>
A	enters assembly-language mnemonics with operands.
D	displays memory in hexadecimal and ASCII.
F	fills memory with constant data.
G	begins execution with optional breakpoints.
I	sets up a standard input File Control Block.
L	lists memory using assembler mnemonics.
M	moves a memory segment from source to destination.
R	reads a program for subsequent testing.

Table 4-2. (continued)

<i>Command Character</i>	<i>Result</i>
S	substitutes memory values.
T	traces program execution.
U	untraced program monitoring.
X	examines and optionally alters the CPU state.

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values, which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. The commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, you can stop execution of DDT by using either a CTRL-C or GO (jump to location 0000H) and save the current memory image by using a SAVE command of the form:

**SAVE n filename. COM**

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks is determined by taking the high-order byte of the address in the TPA and converting this number to decimal. For example, if the highest address in the TPA is 134H, the number of pages is 12H or 18 in decimal. You could type a CTRL-C during the debug run, returning to the CCP level, followed by

**SAVE 18 X.COM**

The memory image is saved as X.COM on the disk and can be directly executed by typing the name X. If further testing is required, the memory image can be recalled by typing

**DDT X.COM**

which reloads the previously saved program from location 100H through page 18, 23FFH. The CPU state is not a part of the COM file; thus, the program must be restarted from the beginning to test it properly.

## 4.2 DDT Commands

The individual commands are detailed below. In each case, the operator must wait for the hyphen prompt character before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel. In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, the numbers are represented by lower-case letters. These numbers are always assumed to be in a hexadecimal radix and from one to four digits in length. Longer numbers are automatically truncated on the right.

Many of the commands operate upon a CPU state that corresponds to the program under test. The CPU state holds the registers of the program being debugged and initially contains zeros for all registers and flags except for the program counter, P, and stack pointer, S, which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded, see the I and R commands.

### 4.2.1 The A (Assembly) Command

DDT allows in-line assembly language to be inserted into the current memory image using the A command, which takes the form:

As

where s is the hexadecimal starting address for the in-line assembly. DDT prompts the console with the address of the next instruction to fill and reads the console, looking for assembly-language mnemonics followed by register references and operands in absolute hexadecimal form. See the *Intel 8080 Assembly Language Reference Card* for a list of mnemonics. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, you can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlaid by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used.

#### 4.2.2 The D (Display) Command

The D command allows you to view the contents of memory in hexadecimal and ASCII formats. The D command takes the forms:

D  
Ds  
Ds,f

In the first form, memory is displayed from the current display address, initially 100H, and continues for 16 display lines. Each display line takes the following form:

aaaa bb cccccccccccccc

where aaaa is the display address in hexadecimal and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are to the right (represented by the sequence of character c) where nongraphic characters are printed as a period. You should note that both upper- and lower-case alphabets are displayed, and will appear as upper-case symbols on a console device that supports only upper-case. Each display line gives the values of 16 bytes of data, with the first line truncated so that the next line begins at an address that is a multiple of 16.

The second form of the D command is similar to the first, except that the display address is first set to address s.

The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pressing the return key.

#### 4.2.3 The F (Fill) Command

The F command takes the form:

Fs,f,c,

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. DDT stores the constant c at address s, increments the value of s and test against f. If s exceeds f, the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

#### 4.2.4 The G (Go) Command

A program is executed using the G command, with up to two optional breakpoint addresses. The G command takes the forms:

G  
Gs  
Gs,b  
Gs,b,c  
G,b  
G,b,c

The first form executes the program at the current value of the program counter in the current machine state, with no breakpoints set. The only way to regain control in DDT is through a RST 7 execution. The current program counter can be viewed by typing an X or XP command.

The second form is similar to the first, except that the program counter in the current machine state is set to address s before execution begins.

The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered.

The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are cleared. The last two forms take the program counter from the current machine state and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. There is no intervention between the starting address and the break address by DDT. If the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

\* d

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. You must specify breakpoints that differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the following commands:

```
G,1234  
G400,400
```

both produce an immediate breakpoint without executing any instructions.

#### 4.2.5 The I (Input) Command

The I command allows you to insert a filename into the default File Control Block (FCB) at 5CH. The FCB created by CP/M for transient programs is placed at this location (see Section 5). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this filename is also used by DDT for reading additional HEX and COM files. The I command takes the forms:

```
Ifilename  
Ifilename.typ
```

If the second form is used and the filetype is either HEX or COM, subsequent R commands can be used to read the pure binary or hex format machine code. Section 4.2.8 gives further details.

#### 4.2.6 The L (List) Command

The L command is used to list assembly-language mnemonics in a particular program region. The L command takes the forms:

```
L  
Ls  
Ls,f
```

The first form lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (G and T commands). Again, long typeouts can be aborted by pressing RETURN during the list process.

#### 4.2.7 The M (Move) Command

The M command allows block movement of program or data areas from one location to another in memory. The M command takes the form:

Ms,f,d

where s is the start address of the move, f is the final address, and d is the destination address. Data is first removed from s to d, and both addresses are incremented. If s exceeds f, the move operation stops; otherwise, the move operation is repeated.

#### 4.2.8 The R (Read) Command

The R command is used in conjunction with the I command to read COM and HEX files from the disk into the transient program area in preparation for the debug run. The R command takes the forms:

R  
Rb

where b is an optional bias address that is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (that is, the first page of memory). If b is omitted, then b = 0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is used for COM files. Note that any number of R commands can be issued following the I command to reread the program under test, assuming the tested program does not destroy the default area at 5CH. Any file specified with the filetype COM is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command,

DDT filename.typ

which initiates the DDT program, equals to the following commands:

DDT  
-I filename.typ  
-R

Whenever the R command is issued, DDT responds with either the error indicator ? (file cannot be opened, or a checksum error occurred in a HEX file) or with a load message. The load message takes the form:

NEXT PC

nnnn pppp

where nnnn is the next address following the loaded program and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

#### 4.2.9 The S (Set) Command

The S command allows memory locations to be examined and optionally altered. The S command takes the form:

Ss

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in memory. If you type a carriage return, the data is not altered. If a byte value is typed, the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until you type either a period or an invalid input value is detected.

#### 4.2.10 The T (Trace) Command

The T command allows selective tracing of program execution for 1 to 65535 program steps. The T command takes the forms:

T

Tn

In the first form, the CPU state is displayed and the next program step is executed. The program terminates immediately, with the termination address displayed as

\*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU stat is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

You should note that program tracing is discontinued at the CP/M interface and resumes after return from CP/M to the program under test. Thus, CP/M functions that access I/O devices, such as the disk drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real-time because DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but commands that use the breakpoint facility (G, T, and U) accomplish the break using an RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

To get control back to DDT during trace, press RETURN rather than executing an RST 7. This ensures that the trace for current instruction is completed before interruption.

#### 4.2.11 The U (Untrace) Command

The U command is identical to the T command, except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535, (0FFFFH) steps to be executed in monitored mode and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

#### 4.2.12 The X (Examine) Command

The X command allows selective display and alteration of the current CPU state for the program under test. The X command takes the forms:

X

Xr

where r is one of the 8080 CPU registers listed in the following table.

Table 4-3. CPU Registers

Register	Meaning	Value
C	Carry flag	(0/1)
Z	Zero flag	(0/1)
M	Minus flag	(0/1)
E	Even parity flag	(0/1)
I	Interdigit carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack pointer	(0-FFFF)
P	Program counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format:

CfZfMfEfI f A = bb B = dddd D = dddd H = dddd S = dddd P = dddd inst

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double-byte quantity corresponding to the register pair. The inst field contains the disassembled instruction, that occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, the flag or register value is not altered. If a value in the proper range is typed, the flag or register value is altered. You should note that BC, DE, and HL are displayed as register pairs. Thus, you must type the entire register pair when B, C, or the BC pair is altered.

### 4.3 Implementation Notes

The organization of DDT allows certain nonessential portions to be overlaid to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the CCP and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus, which, in turn, contains a JMP instruction to the BDOS. Thus, programs that use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process, the DDT program again alters the address field at 6H to include this module, further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a ? in response) and the trace and display (T and X) commands list the inst field of the display in hexadecimal, rather than as a decoded instruction.

## 4.4 A Sample Program

The following example shows an edit, assemble, and debug for a simple program that reads a set of data values and determines the largest value in the set. The largest value is taken from the vector and stored into LARGE at the termination of the program.

A&gt;ED SCAN.ASM

Create source program;  
“*r*” represents carriage return.

```
*I
        ORG      1-00H      ;START OF TRANSIENT
        MVI      B, LEN      ;AREA
        MVI      C, 0       ;LENGTH OF VECTOR TO SCAN
LOOP:   LXI      H, VECT    ;LARGER_RST VALUE SO FAR
        MOV      A, M       ;BASE OF VECTOR
        SUB      C           ;GET VALUE
        JNC      NFOUND    ;LARGER VALUE IN C?
        ;           ;JUMP IF LARGER VALUE NOT
        ;           ;FOUND
        ;           ;NEW LARGEST VALUE, STORE IT TO C
        MOV      C, A       ;TO NEXT ELEMENT
NFOUND: INX      H           ;MORE TO SCAN?
        DCR      B           ;FOR ANOTHER
        JNZ      LOOP
        ;           ;END OF SCAN, STORE C
```

```

        MOV      A, C      ;GET LARGEST VALUE
        STA      LARGE      ;REBOOT
        JMP      0           ;REBOOT

; TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN:    EQU      $-VECT      ;LENGTH
LARGE:   DS      1           ;LARGEST VALUE ON EXIT
END.    ;REBOOT

^-Z
*BOP.  ;REBOOT

        ORG      100H      ;START OF TRANSIENT AREA
        MVI      B,LEN      ;LENGTH OF VECTOR TO SCAN
        MVI      C,0      ;LARGEST VALUE SO FAR
        LXI      H,VECT      ;BASE OF VECTOR
LOOP:   MOV      A,M      ;GET VALUE
        SUB      C          ;LARGER VALUE IN C?
        JNC      NFOUND      ;JUMP IF LARGER VALUE NOT
                           ;FOUND

; NEW LARGEST VALUE, STORE IT TO C
        MOV      C,A
        INX      H          ;TO NEXT ELEMENT
        DCR      B          ;MORE TO SCAN?
        JNZ      LOOP      ;FOR ANOTHER

; END OF SCAN, STORE C
        MOV      A,C      ;GET LARGEST VALUE
        STA      LARGE      ;REBOOT
        JMP      0           ;REBOOT

; TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN:    EQU      $-VECT      ;LENGTH
LARGE:   DS      1           ;LARGEST VALUE ON EXIT
END.    ;REBOOT

```

\*E← End of edit

A>ASM SCAN ↴ Start Assembler

CP/M ASSEMBLER - VER 1.0

0122

002H USE FACTOR

END OF ASSEMBLY    Assembly complete; lock at program listing

A>TYPE SCAN.PRN ↴

Code address    Source program

0100		ORG 100H	; START OF TRANSIENT AREA
0100	0608	MVI B,LEN	; LENGTH OF VECTOR TO SCAN
0102	0E00	MVI C,0	; LARGEST VALUE SO FAR
0104	211901	LXI H,VECT.	; BASE OF VECTOR
0107	7E	LOOP: MOV A,M	; GET VALUE
0108	91	SUB C	; LARGER VALUE IN C?
0109	D20D01	JNC NFOUND	; JUMP IF LARGER VALUE NOT ; FOUND
		;	NEW LARGEST VALUE, STORE IT TO C
010C	4F	MOV C,A	
010D	23	NFOUND: INX H	; TO NEXT ELEMENT
010E	05	DCR B	; MORE TO SCAN?
010F	C20701	JNZ LOOP	; FOR ANOTHER
		;	
		;	END OF SCAN, STORE C
0112	79	MOV A,C	; GET LARGEST VALUE
0113	322101	STA LARGE	
0116	C30000	JMP 0	;REBOOT
Code—data listing ;			
truncated	;	TEST DATA	
0119	0200040305	VECT: DB 2,0,4,3,5,6,1,5	
0008	= Value of LEN	EQU \$-VECT	; LENGTH
0121	equate LARGE:	DS 1	; LARGEST VALUE ON EXIT
0122		END	

A>DDT SCAN,HEX Start debugger using hex format machine code

DDT VER 1.0

NEXT PC

0121 0000

-X Last load address + 1

Next instruction  
to execute at  
PC = 0

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT 7F

-XP Examine registers before debug run

P=0000 100 Change PC to 100

-X Look at registers again

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08

-L100

PC changed

Next instruction  
to execute at PC = 100

0100	MVI	B,08
0102	MVI	C,00
0104	LXI	H,0119
0107	MOV	A,M
0108	SUB	C
0109	JNC	010D
010C	MOV	C,A
010D	INX	H
010E	DCR	B
010F	JNZ	0107
0112	MOV	A,C

Disassembled machine  
code at 100H  
(see source listing  
for comparison)

-L

0113	STA	0121
0116	JMP	0000
0119	STAX	B
011A	NOP	
011B	INR	B
011C	INX	B
011D	DCR	B
011E	MVI	B,01
0120	DCR	B
0121	LXI	D,2200
0124	LXI	H,0200

A little more machine code. Note that program ends at location 116 with a JMP to 0000. Remainder of listing is assembly of data.

-A116 ↗ Enter in-line assembly mode to change the JMP to 0000 into a RST 7, which will cause the program under test to return to DDT if 116H is ever executed.

0116 RST 7

0117 ↗ (Single carriage return stops assemble mode)

-L113 ↗ List code at 113H to check that RST 7 was properly inserted

0113	STA	0121
0116	RST	07 in place of JMP
0117	NOP	
0118	NOP	
0119	STAX	B
011A	NOP	
011B	INR	B
011C	INX	B

-X ↗ Look at registers

COZOMOE0IO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08

-T ↗ Execute Program for one stop. Initial CPU state, before ↗ is executed

COZOMOE0IO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08\*0102  
Automatic breakpoint



CPU state at end of US  
COZ0M0E1I1 A=04 B=0600 D=0000 H=011B S=0100 P=010B SUB C  
-G Run program from current PC until completion (in real-time)

\*0116 breakpoint at 116H, caused by executing RST 7 in machine code.

-X CPU state at end of program  
COZ1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07  
-XP Examine and change program counter

P=0116 100

-X  
COZ1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08  
-TIO

First data element  
 Current largest value  
 Subtract for comparison C

Trace 10 (hexadecimal) steps

```

COZ1MOE1II A=00 B=0800 D=0000 H=0121 S=0100 P=0100 MVI B,08
COZ1MOE1II A=00 B=0000 D=0000 H=0121 S=0100 P=0102 MVI C,00
COZ1MOE1II A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI H,0119
COZ1MOE1II A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
COZ1MOE1II A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
COZOMOE0II A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D
COZOMOE0II A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
COZOMOE0II A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B
COZOMOE0II A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
COZOMOE0II A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
COZOMOE0II A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C
COZ1MOE1II A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
COZ1MOE1II A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
COZ1MOE1II A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR B
COZOMOE1II A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ 0107
COZOMOE1II A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV A,M*0108
-A109/

```

Insert a "hot patch" into  
 the machine code  
 0109 JC 10D to change the  
 JNC to JC  
 010C/

Program should have moved the  
 value from A into C since A>C.  
 Since this code was not executed,  
 it appears that the JNC should  
 have been a JC instruction

Stop DDT so that a version of  
 -GO/ the patched program can be saved

A>SAVE 1 SCAN.COM/ Program resides on first  
 page, so save 1 page.

A>DDT SCAN.COM/ Restart DDT with the save memory  
 image to continue testing

DDT VER 1.0  
 NEXT PC  
 0200 0100

-L100 List some code

```

0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D Previous patch is present in X.COM
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C
-XP

```

P=0100

-T10

Trace to see how patched version operates Data is moved from A to C

```

COZOMOE0IO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
COZOMOE0IO A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00
COZOMOE0IO A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119
COZOMOE0IO A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
COZOMOE0IO A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
COZOMOE0II A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D
COZOMOE0II A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A
COZOMOE0II A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H
COZOMOE0II A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B
COZOMOE0II A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107
COZOMOE0II A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M
COZOMOE0II A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C
C1ZOM1EOIO A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D
C1ZOM1EOIO A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H
C1ZOM1EOIO A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR B
C1ZOMOE111 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ 0107*0107
-X

```

Breakpoint after 16 steps

C1ZOMOE111 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV A,M

-G,108 Run from current PC and breakpoint at 108H

\*0108

-X ↴

Next data item

C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C

-T ↴

Single step for a few cycles

C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C\*0109

-T ↴

COZ0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC 010D\*010C

-X ↴

COZ0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV C,A

-G ↴ Run to completion

\*0116

-X ↴

COZ1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST 07

-S121 ↴ Look at the value of "LARGE"

0121 03 ↴ Wrong value!

0122 00 ↴

0123 22 ↴

0124 21 ↴

0125 00 ↴

0126 02 ↴

0127 7E ↴ - End of the S command

-L100 ↴

0100	MVI	B,08
0102	MVI	C,00
0104	LXI	H,0119
0107	MOV	A,M
0108	SUB	C
0109	JC	010D
010C	MOV	C,A
010D	INX	H
010E	DCR	B
010F	JNZ	0107
0112	MOV	A,C

Review the code

-L ↴

0113	STA	0121
0116	RST	07
0117	NOP	
0118	NOP	
0119	STAX	B
011A	NOP	
011B	INR	B
011C	INX	B
011D	DCR	B
011E	MVI	B,01
0120	DCR	B

-XP ↴

P=0116 100 Reset the PC

-T ↴

Single step, and watch data values

COZ1M0E1II A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI B,08\*0102

-T ↴

COZ1M0E1II A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI C,00\*0104

-T ↴

Count set ↴ Largest set ↴

COZ1M0E1II A=03 B=0800 D=0000 H=0121 S=0100 P=0104 LXI H,0119\*0107

-T ↴

COZ1M0E1I1 A=03 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M\*0108  
-T<sup>✓</sup> Base address of data set

COZ1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C\*0109  
-T<sup>✓</sup> First data item brought to A

COZ0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D\*010C  
-T<sup>✓</sup>

COZ0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A\*010D  
-T<sup>✓</sup>

First data item moved to C correctly

COZ0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H\*010E  
-T<sup>✓</sup>

COZ0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B\*010F  
-T<sup>✓</sup>

COZ0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107\*0107  
-T<sup>✓</sup>

COZ0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M\*0108  
-T<sup>✓</sup>

Second data item brought to A

COZ0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C\*0109  
-T<sup>✓</sup>

Subtract destroys data value that was loaded!

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D\*010D  
-T<sup>✓</sup>

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H\*010E  
-L100<sup>✓</sup>

```

0100 MVI     B,08
0102 MVI     C,00
0104 LXI     H,0119
0107 MOV     A,M
0108 SUB    C ← This should have been a CMP so that register A
0109 JC      010D would not be destroyed.
010C MOV     C,A
010D INX     H
010E DCR     B
010F JNZ     0107
0112 MOV     A,C
-A10B

```

0108 CMP C ↗ Hot patch at 108H changes SUB to CMP

0109

-G ↗ Stop DDT for SAVE

A>SAVE 1 SCAN.COM ↗ Save memory image

A>DDT SCAN.COM ↗ Restart DDT

DDT VER 1.0

NEXT PC

0200 0100

-XP ↗

P=0100

-L116 ↗

0116 RST	07	} Look at code to see if it was properly loaded (long typeout aborted with rubout)
0117 NOP		
0118 NOP		
0119 STAX	B	
011A NOP		

-G,116 ↗ Run from 100H to completion

\*0116

-XC Look at carry (accidental typo)

C1

-X Look at CPU state

C1Z1M0E1I1 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07

-S121 Look at "large"—it appears to be correct.

0121 06

0122 00

0123 22

-GO Stop DDT

A>ED SCAN.ASM Re-edit the source program, and make both changes

\*NSUB

\*OLT

CTRL-Z SUB C ;LARGER VALUE IN C?

\*SSUB1ZCMP1ZOLT CMP D ;LARGER VALUE IN C?

\*

JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND

\*SNC1ZC1ZOLT JC NFOUND ;JUMP IF LARGER VALUE NOT FOUND

\*E

Reassemble, selecting source from disk A

A>ASM SCAN.AAZ ← Hex to disk A

Print to Z (selects no print file)

CP/M ASSEMBLER VER 1.0

0122

002H USE FACTOR

END OF ASSEMBLY

A>DDT SCAN,HEX ↵ Rerun debugger to check changes

DDT VER 1.0

NEXT PC

0121 0000

-L116 ↵

0116 JMP 0000 Check to ensure end is still at 116H

0119 STAX B

011A NOP

011B INR B

-(rubout)

-G100,116 ↵ Go from beginning with breakpoint at end

\*0116 Breakpoint reached

-D121 ↵ Look at "LARGE"

Correct value computed

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 .. '!... W ,#,.,X,

0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..'....),.....,

0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......,.....,

-(rubout) Aborts long typeout

G0 ↵ Stop DDT, debug session complete.

*End of Section 4*

# Section 5

## CP/M 2 System Interface

### 5.1 Introduction

This chapter describes CP/M (release 2) system organization including the structure of memory and system entry points. This section provides the information you need to write programs that operate under CP/M and that use the peripheral and disk I/O facilities of the system.

CP/M is logically divided into four parts, called the Basic Input/Output System (BIOS), the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module that defines the exact low level interface with a particular computer system that is necessary for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided for field reconfiguration of the BIOS to match nearly any hardware environment, see Section 6.

The BIOS and BDOS are logically combined into a single module with a common entry point and referred to as the FDOS. The CCP is a distinct program that uses the FDOS to provide a human-oriented interface with the information that is cataloged on the back-up storage device. The TPA is an area of memory, not used by the FDOS and CCP, where various nonresident operating system commands and user programs are executed. The lower portion of memory is reserved for system information and is detailed in later sections. Memory organization of the CP/M system is shown in Figure 5-1.

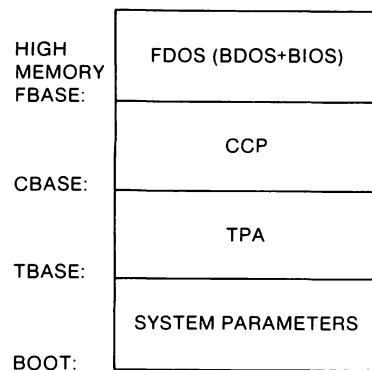


Figure 5-1. CP/M Memory Organization

The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version and are described fully in Section 6. All standard CP/M versions assume  $\text{BOOT} = 0000\text{H}$ , which is the base of random access memory. The machine code found at location  $\text{BOOT}$  performs a system warm start, which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location  $\text{BOOT}$  to return control to CP/M at the command level. Further, the standard versions assume  $\text{TBASE} = \text{BOOT} + 0100\text{H}$ , which is normally location  $0100\text{H}$ . The principal entry point to the FDOS is at location  $\text{BOOT} + 0005\text{H}$  (normally  $0005\text{H}$ ) where a jump to  $\text{FBASE}$  is found. The address field at  $\text{BOOT} + 0006\text{H}$  (normally  $0006\text{H}$ ) contains the value of  $\text{FBASE}$  and can be used to determine the size of available memory, assuming that the CCP is being overlaid by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the following forms:

command  
command file1  
command file1 file2

where command is either a built-in function, such as DIR or TYPE, or the name of a transient command or program. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disk for a file by the name

command.COM

If the file is found, it is assumed to be a memory image of a program that executes in the TPA and thus implicitly originates at TBASE in memory. The CCP loads the COM file from the disk into memory starting at TBASE and can extend up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two File Control Block (FCB) names in the system parameter area. These optional FCBs are in the form necessary to access files through the FDOS and are described in Section 5.2.

The transient program receives control from the CCP and begins execution, using the I/O facilities of the FDOS. The transient program is called from the CCP. Thus, it can simply return to the CCP upon completion of its processing, or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up through FBASE-1 can be used.

The transient program can use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a function number and an information address to CP/M through the FDOS entry point at BOOT + 0005H. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful.

## 5.2 Operating System Call Conventions

This section provides detailed information for performing direct operating system calls from user programs. Many of the functions listed below, however, are accessed more simply through the I/O macro library provided with the MAC macro assembler and listed in the Digital Research manual entitled, *Programmer's Utilities Guide for the CP/M Family of Operating Systems*.

CP/M facilities that are available for access by transient programs fall into two general categories: simple device I/O and disk file I/O. The simple device operations are

- read a console character
- write a console character
- read a sequential character
- write a sequential character
- get or set I/O status
- print console buffer
- interrogate console ready

The following FDOS operations perform disk I/O:

- disk system reset
- drive selection
- file creation
- file close
- directory search
- file delete
- file rename
- random or sequential read
- random or sequential write
- interrogate available disks
- interrogate selected disk
- set DMA address
- set/reset file indicators.

As mentioned above, access to the FDOS functions is accomplished by passing a function number and information address through the primary point at location BOOT+0005H. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL, a zero value is returned when the

function number is out of range. For reasons of compatibility, register A = L and register B = H upon return in all cases. Note that the register passing conventions of CP/M agree with those of the Intel PL/M systems programming language. CP/M functions and their numbers are listed below.

0	System Reset	19	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22	Make File
4	Punch Output	23	Rename File
5	List Output	24	Return Login Vector
6	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Aloc)
9	Print String	28	Write Protect Disk
10	Read Console Buffer	29	Get R/O Vector
11	Get Console Status	30	Set File Attributes
12	Return Version Number	31	Get Addr(DiskParms)
13	Reset Disk System	32	Set/Get User Code
14	Select Disk	33	Read Random
15	Open File	34	Write Random
16	Close File	35	Compute File Size
17	Search for First	36	Set Random Record
18	Search for Next	37	Reset Drive
		40	Write Random with Zero Fill

Functions 28 and 32 should be avoided in application programs to maintain upward compatibility with CP/M.

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight-level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (most transients return to the CCP through a jump to location 0000H) it is large enough to make CP/M system calls because the FDOS switches to a local stack at system entry. For example, the assembly-language program segment below reads characters continuously until an asterisk is encountered, at which time control returns to the CCP, assuming a standard CP/M system with BOOT = 0000H.

```
BDS$    EQU      0005H    ;STANDARD CP/M ENTRY
CONIN   EQU      1        ;CONSOLE INPUT FUNCTION
;
ORG     0100H    ;BASE OF TPA
NEXTC:  MVI      C,CONIN  ;READ NEXT CHARACTER
        CALL    BDS$      ;RETURN CHARACTER IN <A>
        CPI      '*'      ;END OF PROCESSING?
        JNZ      NEXTC    ;LOOP IF NOT
        RET      ;RETURN TO CCP
        END
```

CP/M implements a named file structure on each disk, providing a logical organization that allows any particular file to contain any number of records from completely empty to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk filenames are in three parts: the drive select code, the filename (consisting of one to eight nonblank characters), and the filetype (consisting of zero to three nonblank characters). The filetype names the generic category of a particular file, while the filename distinguishes individual files in each category. The filetypes listed in Table 5-1 name a few generic categories that have been established, although they are somewhat arbitrary.

Table 5-1. CP/M Filetypes

<i>Filetype</i>	<i>Meaning</i>
ASM	Assembler Source
PRN	Printer Listing
HEX	Hex Machine Code
BAS	Basic Source File
INT	Intermediate Code
COM	Command File
PLI	PL/I Source File
REL	Relocatable Module
TEX	TEX Formatter Source
BAK	ED Source Backup
SYM	SID Symbol File
\$\$\$	Temporary File

Source files are treated as a sequence of ASCII characters, where each line of the source file is followed by a carriage return, and line-feed sequence (0DH followed by 0AH). Thus, one 128-byte CP/M record can contain several lines of source text. The end of an ASCII file is denoted by a CTRL-Z character (1AH) or a real end-of-file returned by the CP/M read operation. CTRL-Z characters embedded within machine code files (for example, COM files) are ignored and the end-of-file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. Note, however, that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. The division into extents is discussed in the paragraphs that follow; however, they are not particularly significant for the programmer, because each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with Function 15, DE usually addresses a FCB. Transient programs often use the default FCB area reserved by CP/M at location BOOT + 005CH (normally 005CH) for simple file operations. The basic unit of file information is a 128-byte record used for all file operations. Thus, a default location for disk I/O is provided by CP/M at location BOOT + 0080H (normally 0080H) which is the initial default DMA address. See Function 26.

All directory operations take place in a reserved area that does not affect write buffers as was the case in release 1, with the exception of Search First and Search Next, where compatibility is required.

The FCB data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case when the file is accessed randomly. The default FCB, normally located at 005CH, can be used for random access files, because the three bytes starting at BOOT + 007DH are available for this purpose. Figure 5-2 shows the FCB format with the following fields.

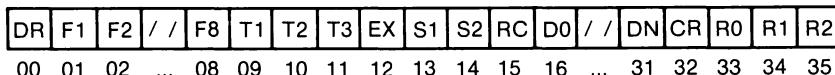


Figure 5-2. File Control Block Format

The following table lists and describes each of the fields in the File Control Block figure.

**Table 5-2. File Control Block Fields**

<i>Field</i>	<i>Definition</i>
dr	drive code (0–16) 0 = use default drive for file 1 = auto disk select drive A, 2 = auto disk select drive B, . . . 16 = auto disk select drive P.
f1...f8	contain the filename in ASCII upper-case, with high bit = 0
t1, t2, t3	contain the filetype in ASCII upper-case, with high bit = 0 t1', t2', and t3' denote the bit of these positions, t1' = 1 =>Read-Only file, t2' = 1 =>SYS file, no DIR list
ex	contains the current extent number, normally set to 00 by the user, but in range 0–31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
rc	record count for extent ex; takes on values from 0–127
d0...dn	filled in by CP/M; reserved for system use
cr	current record to read or write in a sequential file operation; normally set to zero by user
r0, r1, r2	optional random record number in the range 0–65535, with overflow to r2, r0, r1 constitute a 16-bit value with low byte r0, and high byte r1

Each file being accessed through CP/M must have a corresponding FCB, which provides the name and allocation information for all subsequent file operations. When accessing files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB and initialize the cr field. Normally, bytes 1 through 11 are set to the ASCII character values for the filename and filetype, while all other fields are zero.

FCBs are stored in a directory area of the disk, and are brought into central memory before the programmer proceeds with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation, (see the CLOSE command).

The CCP constructs the first 16 bytes of two optional FCBs for a transient by scanning the remainder of the line following the transient name, denoted by file1 and file2 in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location BOOT + 005CH and can be used as is for subsequent file operations. The second FCB occupies the d0...dn portion of the first FCB and must be moved to another area of memory before use. If, for example, the following command line is typed:

```
PROGNAME B:X,ZOT Y,ZAP
```

the file PROGNAME.COM is loaded into the TPA, and the default FCB at BOOT + 005CH is initialized to drive code 2, filename X, and filetype ZOT. The second drive code takes the default value 0, which is placed at BOOT - 006CH, with the filename Y placed into location BOOT + 006DH and filetype ZAP located 8 bytes later at BOOT + 0075H. All remaining fields through cr are set to zero. Note again that it is the programmer's responsibility to move this second filename and filetype to another area, usually a separate file control block, before opening the file that begins at BOOT + 005CH, because the open operation overwrites the second name and type.

If no filenames are specified in the original command, the fields beginning at BOOT + 005DH and BOOT + 006DH contain blanks. In all cases, the CCP translates lower-case alphabetics to upper-case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location  $\text{BOOT} + 0080\text{H}$  is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at  $\text{BOOT} + 0080\text{H}$  is initialized as follows:

$\text{BOOT} + 0080\text{H}:$

+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +A +B +C +D +E  
E " 'B' ':' 'X' ':' 'Z' 'O' 'T' " 'Y' ':' 'Z' 'A' 'P'

where the characters are translated to upper-case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

Individual functions are described in detail in the pages that follow.

#### FUNCTION 0: SYSTEM RESET

Entry Parameters:  
Register C: 00H

The System Reset function returns control to the CP/M operating system at the CCP level. The CCP reinitializes the disk subsystem by selecting and logging-in disk drive A. This function has exactly the same effect as a jump to location  $\text{BOOT}$ .

**FUNCTION 1: CONSOLE INPUT****Entry Parameters:**

Register C: 01H

**Returned Value:**

Register A: ASCII Character

The Console Input function reads the next console character to register A. Graphic characters, along with carriage return, line-feed, and back space (CTRL-H) are echoed to the console. Tab characters, CTRL-I, move the cursor to the next tab stop. A check is made for start/stop scroll, CTRL-S, and start/stop printer echo, CTRL-P. The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

**FUNCTION 2: CONSOLE OUTPUT****Entry Parameters:**

Register C: 02H

Register E: ASCII Character

The ASCII character from register E is sent to the console device. As in Function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

**FUNCTION 3: READER INPUT**

Entry Parameters:

Register C: 03H

Returned Value:

Register A: ASCII Character

The Reader Input function reads the next character from the logical reader into register A. See the IOBYTE definition in Section 6. Control does not return until the character has been read.

**FUNCTION 4: PUNCH OUTPUT**

Entry Parameters:

Register C: 04H

Register E: ASCII Character

The Punch Output function sends the character from register E to the logical punch device.

**FUNCTION 5: LIST OUTPUT****Entry Parameters:**

Register C: 05H

Register E: ASCII Character

The List Output function sends the ASCII character in register E to the logical listing device.

**FUNCTION 6: DIRECT CONSOLE I/O****Entry Parameters:**

Register C: 06H

Register E: 0FFH (input) or  
char (output)**Returned Value:** char or status

Register A:

Direct Console I/O is supported under CP/M for those specialized applications where basic console input and output are required. Use of this function should, in general, be avoided since it bypasses all of the CP/M normal control character functions (for example, CTRL-S and CTRL-P). Programs that perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to Function 6, register E either contains hexadecimal FF, denoting a console input request, or an ASCII character. If the input value is FF, Function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, Function 6 assumes that E contains a valid ASCII character that is sent to the console.

Function 6 must not be used in conjunction with other console I/O functions.

**FUNCTION 7: GET I/O BYTE**

Entry Parameters:  
Register C: 07H

Returned Value:  
Register A: I/O Byte Value

The Get I/O Byte function returns the current value of IOBYTE in register A. See Section 6 for IOBYTE definition.

**FUNCTION 8: SET I/O BYTE**

Entry Parameters:  
Register C: 08H  
Register E: I/O Byte Value

The SET I/O Byte function changes the IOBYTE value to that given in register E.

**FUNCTION 9: PRINT STRING**

Entry Parameters:  
Register C: 09H  
Registers DE: String Address

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a \$ is encountered in the string. Tabs are expanded as in Function 2, and checks are made for start/stop scroll and printer echo.

## FUNCTION 10: READ CONSOLE BUFFER

## Entry Parameters:

Register C: 0AH

Registers DE: Buffer Address

## Returned Value:

Console Characters in Buffer

The Read Buffer function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either input buffer overflows or a carriage return or line-feed is typed. The Read Buffer takes the form:

DE: +0 +1 +2 +3 +4 +5 +6 +7 +8 . . . +n

mx nc c1 c2 c3 c4 c5 c6 c7 . . . ??

where mx is the maximum number of characters that the buffer will hold, 1 to 255, and nc is the number of characters read (set by FDOS upon return) followed by the characters read from the console. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the above figure. A number of control functions, summarized in Table 5-3, are recognized during line editing.

Table 5-3. Edit Control Characters

<i>Character</i>	<i>Edit Control Function</i>
RUB / DEL	removes and echoes the last character
CTRL-C	reboots when at the beginning of line
CTRL-E	causes physical end of line
CTRL-H	backspaces one character position
CTRL-J	(line-feed) terminates input line

Table 5-3. (continued)

Character	Edit Control Function
CTRL - M	(return) terminates input line
CTRL - R	retypes the current line after new line
CTRL - U	removes current line
CTRL - X	same as CTRL-U

The user should also note that certain functions that return the carriage to the leftmost position (for example, CTRL-X) do so only to the column position where the prompt ended. In earlier releases, the carriage returned to the extreme left margin. This convention makes operator data input and line correction more legible.

#### FUNCTION 11: GET CONSOLE STATUS

##### Entry Parameters:

Register C: 0BH

##### Returned Value:

Register A: Console Status

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value OFFH is returned in register A. Otherwise a 00H value is returned.

**FUNCTION 12: RETURN VERSION NUMBER****Entry Parameters:**

Register C: 0CH

**Returned Value:** Version Number

Registers HL:

Function 12 provides information that allows version independent programming. A two-byte value is returned, with H = 00 designating the CP/M release (H = 01 for MP/M) and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21,22, through 2F. Using Function 12, for example, the user can write application programs that provide both sequential and random access functions.

**FUNCTION 13: RESET DISK SYSTEM****Entry Parameters:**

Register C: 0DH

The Reset Disk function is used to programmatically restore the file system to a reset state where all disks are set to Read-Write. See functions 28 and 29, only disk drive A is selected, and the default DMA address is reset to BOOT + 0080H. This function can be used, for example, by an application program that requires a disk change without a system reboot.

**FUNCTION 14: SELECT DISK****Entry Parameters:**

Register C: 0EH

Register E: Selected Disk

The Select Disk function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15, corresponding to drive P in a full 16 drive system. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or disk system reset operation. If the disk medium is changed while it is on-line, the drive automatically goes to a Read-Only status in a standard CP/M environment, see Function 28. FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16 ignore the selected default drive and directly reference drives A through P.

**FUNCTION 15: OPEN FILE****Entry Parameters:**

Register C: 0FH

Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

The Open File operation is used to activate a file that currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed) where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included, and bytes ex and s2 of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of FCB, thus allowing access to the files through subsequent read and write operations. The user should note that an existing file must not be accessed until a successful open operation is completed. Upon return, the open function returns a directory code with the value 0 through 3 if the open was successful or OFFH (255 decimal) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated. Note that the current record, (cr) must be zeroed by the program if the file is to be accessed sequentially from the first record.

#### FUNCTION 16: CLOSE FILE

##### Entry Parameters:

Register C: 10H

Registers DE: FCB Address

##### Returned Value:

Register A: Directory Code

The Close File function performs the inverse of the Open File function. Given that the FCB addressed by DE has been previously activated through an open or make function, the close function permanently records the new FCB in the reference disk directory see functions 15 and 22. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a OFFH (255 decimal) is returned if the filename cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, the close operation is necessary to record the new directory information permanently.

## FUNCTION 17: SEARCH FOR FIRST

## Entry Parameters:

Register C: 11H  
Registers DE: FCB Address

## Returned Value:

Register A: Directory Code

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A \* 32 (that is, rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, the auto disk select function is disabled and the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

**FUNCTION 18: SEARCH FOR NEXT**

Entry Parameters:

Register C: 12H

Returned Value:

Register A: Directory Code

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to Function 17, Function 18 returns the decimal value 255 in A when no more directory items match.

**FUNCTION 19: DELETE FILE**

Entry Parameters:

Register C: 13H

Registers DE: FCB Address

Returned Value:

Register A: Directory Code

The Delete File function removes files that match the FCB addressed by DE. The filename and type may contain ambiguous references (that is, question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a value in the range 0 to 3 returned.

**FUNCTION 20: READ SEQUENTIAL****Entry Parameters:**

Register C: 14H

Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

Given that the FCB addressed by DE has been activated through an Open or Make function, the Read Sequential function reads the next 128-byte record from the file into memory at the current DMA address. The record is read from position cr of the extent, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next read operation. The value 00H is returned in the A register if the read operation was successful, while a nonzero value is returned if no data exist at the next record position (for example, end-of-file occurs).

**FUNCTION 21: WRITE SEQUENTIAL****Entry Parameters:**

Register C: 15H

Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

Given that the FCB addressed by DE has been activated through an Open or Make function, the Write Sequential function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position cr of the file, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case, newly written records overlay those that already exist in the file. Register A = 00H upon return from a successful write operation, while a nonzero value indicates an unsuccessful write caused by a full disk.

**FUNCTION 22: MAKE FILE****Entry Parameters:**

Register C: 16H  
Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

The Make File operation is similar to the Open File operation except that the FCB must name a file that does not exist in the currently referenced disk directory (that is, the one named explicitly by a nonzero dr code or the default disk if dr is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate filenames occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and OFFH (255 decimal) if no more directory space is available. The Make function has the side effect of activating the FCB and thus a subsequent open is not necessary.

**FUNCTION 23: RENAME FILE****Entry Parameters:**

Register C: 17H  
Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code dr at position 0 is used to select the drive, while the drive code for the new filename at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful and OFFH (255 decimal) if the first filename could not be found in the directory scan.

**FUNCTION 24: RETURN LOG-IN VECTOR**

Entry Parameters:

Register C: 18H

Returned Value:

Registers HL: Log-in Vector

The log-in vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A and the high-order bit of H corresponds to the sixteenth drive, labeled P. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit disk drive selection or an implicit drive select caused by a file operation that specified a nonzero dr field. The user should note that compatibility is maintained with earlier releases, because registers A and L contain the same values upon return.

**FUNCTION 25: RETURN CURRENT DISK**

Entry Parameters:

Register C: 19H

Returned Value:

Register A: Current Disk

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

**FUNCTION 26: SET DMA ADDRESS****Entry Parameters:**

Register C: 1AH  
Registers DE: DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (that is, the data is transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128-byte data record resides before a disk write and after a disk read. Upon cold start, warm start, or disk system reset, the DMA address is automatically set to BOOT+0080H. The Set DMA function can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

**FUNCTION 27: GET ADDR (ALLOC)****Entry Parameters:**

Register C: 1BH

**Returned Value:**

Registers HL: ALLOC Address

An allocation vector is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected disk drive. However, the allocation information might be invalid if the selected disk has been marked Read-Only. Although this function is not normally used by application programs, additional details of the allocation vector are found in Section 6.

**FUNCTION 28: WRITE PROTECT DISK****Entry Parameters:**

Register C: 1CH

The Write Protect Disk function provides temporary write protection for the currently selected disk. Any attempt to write to the disk before the next cold or warm start operation produces the message:

**BDO\$ ERR on d:R/O****FUNCTION 29: GET READ-ONLY VECTOR****Entry Parameters:**

Register C: 1DH

**Returned Value:**

Registers HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL, which indicates drives that have the temporary Read-Only bit set. As in Function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to Function 28 or by the automatic software mechanisms within CP/M that detect changed disks.

**FUNCTION 30: SET FILE ATTRIBUTES****Entry Parameters:**

Register C: 1EH

Registers DE: FCB Address

**Returned Value:**

Register A: Directory Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes ( $t1'$  and  $t2'$ ) can be set or reset. The DE pair addresses an unambiguous filename with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. Indicators  $f1'$  through  $f4'$  are not currently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators  $f5'$  through  $f8'$  and  $t3'$  are reserved for future system expansion.

**FUNCTION 31: GET ADDR (DISK PARMS)****Entry Parameters:**

Register C: 1FH

**Returned Value:**

Registers HL: DPB Address

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

**FUNCTION 32: SET/GET USER CODE****Entry Parameters:**

Register C: 20H  
Register E: OFFH (get) or  
User Code (set)

**Returned Value:**

Register A: Current Code or  
(no value)

An application program can change or interrogate the currently active user number by calling Function 32. If register E = OFFH, the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not OFFH, the current user number is changed to the value of E, modulo 16.

**FUNCTION 33: READ RANDOM****Entry Parameters:**

Register C: 21H

**Returned Value:**

Register A: Return Code

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). The user should note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (Function 35). Byte r2 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, the r0, r1 byte pair is treated as a double-byte, or word value, that contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent might or might not contain any allocated data, this ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (r0, r1), and the BDOS is called to read the record.

Upon return from the call, register A either contains an error code, as listed below, or the value 00, indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. However, note that, in this case, the last randomly read record will be reread as one switches from random mode to sequential read and the last record will be rewritten as one switches to a sequential write operation. The user can simply advance the random record position following each random read or write to obtain the effect of sequential I/O operation.

Error codes returned in register A following a random read are listed below.

- 01 reading unwritten data
- 02 (not returned in random mode)
- 03 cannot close current extent
- 04 seek to unwritten extent
- 05 (not returned in read mode)
- 06 seek past physical end of disk

Error codes 01 and 04 occur when a random read operation accesses a data block that has not been previously written or an extent that has not been created, which are equivalent conditions. Error code 03 does not normally occur under proper system operation. If it does, it can be cleared by simply rereading or reopening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is nonzero under the current 2.0 release. Normally, nonzero return codes can be treated as missing data, with zero return codes indicating operation complete.

**FUNCTION 34: WRITE RANDOM****Entry Parameters:**

Register C: 22H

Registers DE: FCB Address

**Returned Value:**

Register A: Return Code

The Write Random operation is initiated similarly to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block that is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the FCB are set to correspond to the random record that is being written. Again, sequential read or write operations can begin following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created as a result of directory overflow.

**FUNCTION 35: COMPUTE FILE SIZE****Entry Parameters:**

Register C: 23H

Registers DE: FCB Address

**Returned Value:**

Random Record Field Set

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous filename that is used in the directory scan. Upon return, the random record bytes contain the virtual file size, which is, in effect, the record address of the record following the end of the file. Following a call to Function 35, if the high record byte r2 is 01, the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value as before (r0 is the least significant byte), which is the file size.

Data can be appended to the end of an existing file by simply calling Function 35 to set the random record position to the end-of-file and then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode and holes exist in the allocation, the file might contain fewer records than the size indicates. For example, if only the last record of an 8-megabyte file is written in random mode (that is, record number 65535), the virtual size is 65536 records, although only one block of data is actually allocated.

**FUNCTION 36: SET RANDOM RECORD****Entry Parameters:**

Register C: 24H

Registers DE: FCB Address

**Returned Value:**

Random Record Field Set

The Set Random Record function causes the BDOS automatically to produce the random record position from a file that has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary initially to read and scan a sequential file to extract the positions of various key fields. As each key is encountered, Function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, the user can move instantly to a particular keyed record by performing a random read, using the corresponding random record number that was saved earlier. The scheme is easily generalized for variable record lengths, because the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of Function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, Function 36 is called, which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

**FUNCTION 37: RESET DRIVE****Entry Parameters:**

Register C: 25H

Registers DE: Drive Vector

**Returned Value:**

Register A: 00H

The Reset Drive function allows resetting of specified drives. The passed parameter is a 16-bit vector of drives to be reset; the least significant bit is drive A:.

To maintain compatibility with MP/M, CP/M returns a zero value.

**FUNCTION 40: WRITE RANDOM WITH ZERO FILL****Entry Parameters:**

Register C: 28H

Registers DE: FCB Address

**Returned Value:**

Register A: Return Code

The Write With Zero Fill operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data is written.

### 5.3 A Sample File-to-File Copy Program

The following program provides a relatively simple example of file operations. The program source file is created as COPY.ASM using the CP/M ED program and then assembled using ASM or MAC, resulting in a HEX file. The LOAD program is used to produce a COPY.COM file that executes directly under the CCP. The program begins by setting the stack pointer to a local area and proceeds to move the second name from the default area at 006CH to a 33-byte File Control Block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCBs are ready for processing, because the SFCB at 005CH is properly set up by the CCP upon entry to the COPY program. That is, the first name is placed into the default FCB, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file, and creating the destination file. If all this is successful, the program loops at the label COPY until each record is read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

```

;      SAMPLE file-to-file COPY PROGRAM
;
;      at the CCP level, the command
;
;      COPY a:x,y b:u,v
;
;      copies the file named x,y from drive
;      a to a file named u,v. on drive b.
;
0000 =          boot    equ 0000h    ;system reboot
0005 =          bdos   equ 0005h    ;bdos entry point
005c =          fcbl    equ 005ch    ;first file name
005c =          sfcb    equ fcbl     ;source fcb
006c =          fcb2    equ 006ch    ;second file name
0080 =          dbuff   equ 0080h    ;default buffer
0100 =          tpa     equ 0100h    ;beginning of tpa
;
0009 =          printf   equ 9        ;print buffer func#
000f =          openf   equ 15       ;open file func#
0010 =          closef   equ 16       ;close file func#
0013 =          deletef  equ 19       ;delete file func#
0014 =          readf   equ 20       ;sequential read
0015 =          writef   equ 21       ;sequential write

```

```

0016 =           makef    equ 22      ;make file func#
;
0100             org tpa       ;beginning of tpa
0100 311b02      lxi sp,stack ;local stack
;
;          move second file name to dfcb
0103 0e10        mvi c,16     ;half an fcb
0105 116c00      lxi d,dfcb2  ;source of move
0108 21da01      lxi h,dfcb   ;destination fcb
010b 1a          mfcfb:   ldx d      ;source fcb
010c 13          inx d      ;ready next
010d 77          mov m,a    ;dest fcb
010e 23          inx h      ;ready next
010f 0d          dcr c      ;count 16,,0
0110 c10b01      jnz mfcfb  ;loop 16 times
;
;          name has been removed, zero cr
0113 af          xra a      ;a = 00h
0114 32fa01      sta dfcbcr ;current rec = 0
;
;          source and destination fcb's ready
;
0117 115c00      lxi d,dfcb  ;source file
011a cd6901      call open   ;error if 255
011d 118701      lxi d,notfile ;ready message
0120 3c          inr a      ;255 becomes 0
0121 cc6101      cz finis   ;done if no file
;
;          source file open, prep destination
0124 11da01      lxi d,dfcb  ;destination
0127 cd7301      call delete ;remove if present
;
;          source file open, dest file open
012a 11da01      lxi d,dfcb  ;destination
012d cd8201      call make   ;create the file
0130 119601      lxi d,notdir ;ready message
0133 3c          inr a      ;255 becomes 0
0134 cc6101      cz finis   ;done if no dir space
;
;          source file open, dest file open
;          copy until end of file on source
;
;
```

```

0137 115c00      COPY:    lxi d,dfcb   ;source
013a cd7801      call read   ;read next record
013d b7          ora a      ;end of file?
013e c25101      jnz eofile ;skip write if so
;
;       not end of file, write the record
0141 11da01      lix d,dfcb   ;destination
0144 cd7d01      call write  ;write record
0147 11a901      lxi d,space ;ready message
014a b7          ora a      ;00 if write ok
014b c46101      cnz finis ;end if so
014e c33701      jmp copy   ;loop until eof
;
eofile:  ;end of file, close destination
0151 11da01      lxi d,dfcb   ;destination
0154 cd6e01      call close  ;255 if error
0157 21bb01      lxi h,wrrprot ;ready message
015a 3c          inr a      ;255 becomes 00
015b cc6101      cz finis   ;shouldn't happen
;
;       copy operation complete, end
015e 11cc01      lxi d,normal ;ready message
;
finis  ;write message given by de, reboot
0161 0e09          mvi c,printf
0163 cd5000      call bdos   ;write message
0166 c30000      jmp boot   ;reboot system
;
;       system interface subroutines
;       (all return directly from bdos)
;
0169 0e0f          open:    mvi c,openf
016b c30500      jmp bdos
;
016e 0e10          close:   mvi c,closef
0170 c30500      jmp bdos
;
0173 0e13          delete:  mvi c,deletef
0175 c30500      jmp bdos
;
```

```

0178 0e14      read:   mvi c,readf
017a c30500    jmp bdos
;
017d 0e15      write:  mvi c,writef
017f c30500    jmp bdos
;
0182 0e16      make:   mvi c,makef
0184 c30500    jmp bdos
;
;      console messages
0187 6e6f20f   nofile: db 'no source file$'
0196 6e6f209   nodir:  db 'no directory space$'
01a9 6f7574f   space:  db 'out of dat space$'
01bb 7772695   wrprot: db 'write Protected?$'
01cc 636f700   normal: db 'COPY complete$'
;
;      data areas
01da           dfcb:   ds 33      ;destination fcb
01fa           dfcber: equ dfcb+32 ;current record
;
01fb           stack:  ds 32      ;16 level stack
;
021b           end

```

Note that there are several simplifications in this particular program. First, there are no checks for invalid filenames that could contain ambiguous references. This situation could be detected by scanning the 32-byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the filenames have been included (check locations 005DH and 006DH for nonblank ASCII characters). Finally, a check should be made to ensure that the source and destination filenames are different. An improvement in speed could be obtained by buffering more data on each read operation. One could, for example, determine the size of memory by fetching FBASE from location 0006H and using the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128-byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

## 5.4 A Sample File Dump Utility

The following file dump program is slightly more complex than the simple copy program given in the previous section. The dump program reads an input file, specified in the CCP command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform a warm start at the end of processing.

```
;DUMP program reads input file and displays
;hex data
;
0100          org 100h
0005 =        bdos equ 0005h = ;bdos entry point
0001 =        cons equ 1      ;read console
0002 =        typef equ 2     ;type function
0009 =        printf equ 9    ;buffer print entry
000b =        brkf equ 11   ;break key function
;                         ;(true if char
000f =        openf equ 15   ;file open
0014 =        readf equ 20   ;read function
;
005c =        fcb   equ 5ch   ;file control block
;                         ;address
0080 =        buff  equ 80h   ;input disk buffer
;                         ;address
;
;                         ;non graphic characters
000d =        cr    equ 0dh   ;carriage return
000a =        lf    equ 0ah   ;line feed
;
;                         ;file control block definitions
005c =        fcbdn equ fcb+0 ;disk name
005d =        fcfn equ fcb+1 ;file name
0065 =        fcbft equ fcb+9 ;disk file type (3
;                           ;characters)
0068 =        fcbrl equ fcb+12 ;file's current reel
;                           ;number
006b =        fcbrc equ fcb+15 ;file's record count (0 to
;                           ;128)128)
```

```

007c =           fcbcr'  equ  fcb+32  ;current (next) record
007d =           fcbin   equ  fcb+33  ;inumber (0
;                               ;ifcb length
;                               ;set up stack
0100 210000      lxi  h,0
0103 39          dad  sp
;                               ;entry stack pointer in hl from the CCP
0104 221502      shld  oldsp
;                               ;set sp to local stack area (restored at
;                               ;finis)
0107 315702      lxi  sp,stktop
;                               ;read and print successive buffers
010a cdc101      call  setup    ;set up input file
010d feff        cpi   255     ;255 if file not present
010f c21b01      jnz   openok  ;skip if open is ok
;
;                               ;file not there, give error message and
;                               ;return
0112 11f301      lxi  d,opnmss
0115 cd9c01      call  err
0118 c35101      jmp   finis   ;to return
;
openok:          ;open operation ok, set buffer index to
;end
011b 3e80          mvi  a,80h
011d 321302      sta   ibp    ;set buffer pointer to 80h
;                               ;hl contains next address to print
0120 210000      lxi  h,0    ;start with 0000
;
sloop:           ;loop:
0123 e5          push  h     ;save line position
0124 cda201      call  gnb
0127 e1          pop   h     ;recall line position
0138 da5101      jc    finis  ;carry set by gnb if end
;file
012b 47          mov   b,a
;                               ;print hex values
;                               ;check for line fold
012c 7d          mov   a,l

```

```

012d e60f      ani 0fh      ;check low 4 bits
012f c24401
;      jnz nonum
0132 cd7201
;      print line number
;      call crlf
;      check for break key
0135 cd5901
;      call break
;      accum lsb = 1 if character ready
0138 0f          rrc           ;into carry
0139 da5101
;      jc finis    ;don't print any more
;      mov a,h
013c 7c          call phex
013d cd8f01
0140 7d          mov a,l
0141 cd8f01
nonum
0144 23          inx h        ;to next line number
0145 3e20          mvi a,''
0147 cd6501          call pchar
014a 78          mov a,b
014b cd8f01          call phex
014e c32301          jmp $loop
;
finis
;      end of dump, return to cco
;      (note that a jmp to 0000h reboots)
0151 cd7201          call crif
0154 2a1502          lhld oldsp
0157 f9          sphl
;
stack pointer contains CCP's stack
location
0158 c9          ret           ;to the CCP
;
;
subroutines
;
break:  ;check break key (actually any key will
;do)
push h! push d! push b! environment
;saved
mvi c,brkf

```

```

015e cd0500          call bdos
0161 c1d1e1          pop b! pop d! pop h; environment
                      restored
0164 c9              ret
;
pchar:  ;print a character
0165 e5d5c5          push h! push d! push b; saved
0168 0e02            mvi c, typef
016a 5f              mov e,a
016b cd0500          call bdos
016e c1d1e1          pop b! pop d! pop h; restored
0171 c9              ret
;
crlf
0172 3e0d            mvi a,cr
0174 cd6501          call pchar
0177 3e0a            mvi a,lf
0179 cd6501          call pchar
017c c9              ret
;
;
pnlb:   ;print nibble in reg a
017d e60f            ani ofh      ;low 4 bits
017f fe0a            cpi 10
0181 d28901          jnc p10
;
0184 c630            less than or equal to 9
0186 c38b01          adi '0'
                jmp prn
;
;
p10:    ; greater or equal to 10
0189 c637            adi 'a' - 10
018b cd6501          call pchar
018e c9              ret
;
phex:   ;print hex char in reg a
018f f5              pushpsw
0190 0f              rrc
0191 0f              rrc
0192 0f              rrc
0193 0f              rrc
0194 cd7d01          call pnib    ;print nibble

```

```

0197 f1          POP PSW
0198 cd7d01      call pnpip
019b c9          ret
;
err:    ;print error message
;      d/e addresses message ending with "$"
019c 0e09      mvi c,printf ;print buffer
                ;function
019e cd0500      call bdos
01a1 c9          ret
;
;
snb:    ;set next byte
01a2 3a1302      lda ibp
01a5 fe80          cpi 80h
01a7 c2b301      jnz s0
;
;      read another buffer
;
;
01aa cdce01      call diskr
01ad b7          ora a      ;zero value if read ok
01ae cab301      jz s0      ;for another byte
;
;      end of data, return with carry set for eof
01b1 37          stc
01b2 c9          ret
;
;
s0:    ;read the byte at buff+res a
01b3 5f          mov e,a      ;Is byte of buffer index
01b4 1600          mvi d,0      ;double precision
;
01b6 3c          inr a      ;index=index+1
01b7 321302      sta ibp      ;back to memory
;
;      pointer is incremented
;      save the current file address
01ba 218000      lxi h,buff
01bd 19          dad d      ;absolute character address is in hl
;
01be 7e          mov a,m      ;byte is in the accumulator
;
01bf b7          ora a      ;reset carry bit
01c0 c9          ret
;

```

```

        setup:  ;set up file
        ;      open the file for input
        01c1 af          xra a      ;zero to accum
        01c2 327c00       sta fcbcr ;clear current record
        ;
        01c5 115c00      lxi d,fcb
        01c8 0e0f          mvi c,openf
        01ca cd0500          call bdos
        ;
        01cd c9          ;255 in accum if open error
        ret
        ;
        diskr: ;read disk file record
        push h! push d! push b
        lxi d,fcb
        mvi c,readf
        call bdos
        pop b! pop d! pop h
        ret
        ;
        ;      fixed message area
        01ce e5d5c5      signon: db 'file dump version 2.0$'
        01d1 115c00      opnmssg: db cr,lf,'no input file present on
                           disk$'
        ;
        01d4 0e14          ;variable area
        01d6 cd0500      ibp:   ds 2      ;input buffer pointer
        01d9 c1d1e1      oldsp: ds 2      ;entry sp value from CCP
        ;
        01dc c9          ;
        01dd 46494c0      0213
        01f3 0d0a4e0      0215
        ;
        0217              ;      stack area
        stktop:           ;      ds 64      ;reserve 32 level stack
        ;
        0257              end

```

## 5.5 A Sample Random Access Program

This section concludes with an extensive example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. When a program has been created, assembled, and placed into a file labeled RANDOM.COM, the CCP level command

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, followed by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the CCP. In the interest of brevity, the only error message is

error, try again.

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label ready where the individual commands are interpreted. The DFBC at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called readc. This particular program shows the elements of random access processing, and can be used as the basis for further program development.

## Sample Random Access Program for CP/M 2.0

```

0100           org   100h    ;base of tpa
                ;
0000 =         reboot  equ   0000h    ;system reboot
0005 =         bdos    equ   0005h    ;bdos entry point
                ;
0001 =         coninP  equ   1      ;console input function
0002 =         conout   equ   2      ;console output function
0009 =         Pstring  equ   9      ;print string until '$'
000a =         Rstring  equ   10     ;read console buffer
000c =         version   equ   12     ;return version number
000f =         openf    equ   15     ;file open function
0010 =         closef   equ   16     ;close function
0016 =         makef    equ   22     ;make file function
0021 =         readr    equ   33     ;read random
0022 =         writer   equ   34     ;write random
                ;
005c =         fcb     equ   005ch   ;default file control
                ;block
007d =         ranrec   equ   fcb+33  ;random record position
007f =         ranovf   equ   fcb+35  ;high order (overflow)
                ;byte
0080 =         buff    equ   0080h   ;buffer address
                ;
000d =         cr      equ   0dh    ;carriage return
000a =         lf      equ   0ah    ;line feed
                ;

```

## Load SP, Set-Up File for Random Access

```

0100 31bc00          lxi   sp,stack
                ;
                ;version 2.0
0103 0e0c          mvi   c,version
0105 cd0500          call  bdos
0108 fe20          cpi   20h    ;version 2.0 or better?
010a d21600          jnc   versok
                ;bad version, message and go back

```

```

010d 111b00      lxi    d,badver
0110 cdda00      call   print
0113 c30000      jmp   reboot
;
versok:
;      correct version for random access
0116 0e0f      mvi    c,openf  ;open default fcb
0118 115c00      lxi    d,fcb
011b cd 0500      call   bdos
011e 3c          inr    a        ;err 255 becomes zero
011f c23700      jnz   ready
;
;      cannot open file, so create it
0122 0e16      mvi    c,makef
0124 115c00      lxi    d,fcb
0127 cd0500      call   bdos
012a 3c          inr    a        ;err 255 becomes zero
012b c23700      jnz   ready
;
;      cannot create file, directory full
012e 113a00      lxi    d,nospace
0131 cdda00      call   print
0134 c30000      jmp   reboot  ;back to CCP

```

### Loop Back to Ready After Each Command

```

;
ready:
;      file is ready for processing
;
0137 cde500      call   readcom ;read next command
013a 227d00      shld  ranrec ;store input record#
013d 217f00      lxi   h,ranovf
0140 3600      mvi   m,0      ;clear high byte if set
0142 fe51      cpi   'Q'      ;quit?
0144 c25600      ;      jnz   notq
;      quit processing, close file
0147 0e10      mvi   c,closef
0149 115c00      lxi   d,fcb
014c cd0500      call   bdos

```

```

014f 3c          inr    a      ;err 255 becomes 0
0150 cab900      jz     error   ;error message, retry
0153 c30000      jmp    reboot  ;back to CCP
;
```

**End of Quit Command, Process Write**

```

notq:
;      not the quit command, random write?
0156 fe57          cpi    'W'
0158 c28900         jnz    notw
;

;      this is a random write, fill buffer until cr
015b 114d00         lxi    d,datmsg
015e cdda00         call   print   ;data prompt
0161 0e7f           mvi    c,127  ;up to 127 characters
0163 218000         lxi    h,buf   ;destination
rloop:  ;read next character to buff
0166 c5             push   b      ;save counter
0167 e5             push   h      ;next destination
0168 cdc200         call   getch   ;character to a
016b e1             pop    h      ;restore counter
016c c1             pop    b      ;restore next to fill
016d fe0d           cpi    cr     ;end of line?
016f ca7800         jz     erloop
;

;      not end, store character
0172 77             mov    m,a
0173 23             inx    h      ;next to fill
0174 0d             dcr    c      ;counter goes down
0175 c26600         jnz    rloop  ;end of buffer?
erloop: ;end of read loop, store 00
0178 3600           mvi    m,0
;

;      write the record to selected record number
017a 0e22           mvi    c,writer
017c 115c00         lxi    d,fcb
017c cd0500         call   bdos
0182 b7             ora    a      ;error code zero?
0183 c2b900         jnz    error  ;message if not
0186 c33700         jmp    ready  ;for another record
;
```

## End of Write Command, Process Read

```

notw:
;      not a write command, read record?
0189 fe52          cpi    'R'
018t c2b900         jnz    error   ;skip if not
;
;      read random record
018e 0e21          mvi    c,readr
0190 115c00         lxi    d,fcb
0193 cd0500         call   bdos
0196 b7            ora    a       ;return code 00?
0197 c2b900         jnz    error
;
;      read was successful, write to console
019a cdcf00         call   crlf   ;new line
019d 0e80          mvi    c,128   ;max 128 characters
019f 218000         lxi    h,buff  ;next to get
wloop:
01a2 7e            mov    a,m   ;next character
01a3 23            inx    h     ;next to get
01a4 e67f          ani    7fh   ;mask parity
01a6 ca3700         jz    ready  ;for another command
;if 00
01a9 c5            push   b     ;save counter
01aa e5            push   h     ;save next to get
01ab fe20          cpi    ''    ;graphic?
01ad d4c800         cnc    putchr ;skip output if not
01b0 e1            pop    h
01b1 c1            pop    b
01b2 0d            dcr    c     ;count=count-1
01b3 c2a200         jnz    wloop
01b6 c33700         jmp    ready

```

## End of Read Command, All Errors End Up Here

```

;
error:
01b9 115900         lxi    d,errmsg
01bc cdda00         call   print
01bf c33700         jmp    ready
;
```

## Utility Subroutines for Console I/O

```

setchr:           ;read next console character to a
    01c2 0e01      mvi   c,coninP
    01c4 cd0500    call  bdos
    01c7 c9        ret

;

putchr:          ;write character from a to console
    01c8 0e02      mvi   c,conout
    01ca 5f        mov   e,a      ;character to send
    01cb cd0500    call  bdos      ;send character
    01ce c9        ret

;

crlf:            ;send carriage return line feed
    01cf 3e0d      mvi   a,cr      ;carriage return
    01d1 cdc800    call  putchr
    01d4 3e0a      mvi   a,lf      ;line feed
    01d6 cdc800    call  putchr
    01d9 c9        ret

;

print:           ;print the buffer addressed by de until $
    01da d5        push  d
    01db cdcf00    call  crlf
    01de d1        pop   d      ;new line
    01df 0e09      mvi   c,pstring
    01e0 cd0500    call  bdos      ;print the string
    01e4 c9        ret

;

readcom:          ;read the next command line to the conbuf
    01e5 116b00    lxi   d,prompt
    01e8 cddaa00    call  print      ;command?
    01eb 0e0a      mvi   c,rstring
    01ed 117a00    lxi   d,conbuf
    01f0 cd0500    call  bdos      ;read command line
    ;           command line is present, scan it
    01f3 210000    lxi   h,0      ;start with 0000
    01f6 117c00    lxi   d,conlin  ;command line

```

```

01f9 1a      readc:  ldx    d      ;next command
              inx    d      ;character
01fa 13      inx    d      ;to next command
01fb b7      ora    a      ;position
              ora    a      ;cannot be end of
              ora    a      ;command
01fc c8      r2
              ;      not zero, numeric?
01fd d630      sui    '0'    ;carry if numeric
01ff fe0a      cpi    10     ;carry if numeric
0201 d21300      jnc    endrd
              ;      add-in next digit
0204 29      dad    h      ;*2
0205 4d      mov    c,l
0206 44      mov    b,h    ;bc = value * 2
0207 29      dad    h      ;*4
0208 29      dad    h      ;*8
0209 09      dad    b      ;*2 + *8 = *10
020a 85      add    l      ;*digit
020b 6f      mov    l,a
020c d2f900      jnc    readc   ;for another char
020f 24      inr    h      ;overflow
0210 c3f900      jmp    readc   ;for another char
              endrd:
              ;      end of read, restore value in a
0213 c630      adi    '0'    ;command
0215 fe61      cpi    'a'    ;translate case?
0217 d8      rc
              ;      lower case, mask lower case bits
0218 e65f      ani    101$1111b
021a c9      ret
              ;

```

**String Data Area for Console Messages**

```

badver:
021b 536f79      db      'sorry, you need CP/M version 2$'
nospace:
023a 4e6f29      db      'no directory space$'
datmsg:

```

```

024d 547970          db      'type data: $'
                    errmsg: db      'error, try again.$'
                    prompt: db      'next command? $'
                    ;

```

### Fixed and Variable Data Area

```

027a 21          conbuf: db      conlen      ;length of console buffer
027b          consiz: ds      1           ;resulting size after read
027c          conlin: ds      32          ;length 32 buffer
0021 =          conlen  equ     $-consiz
                  ;
029c          stack:  ds      32          ;16 level stack
02bc          end

```

Major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting to arbitrary fields within the record. A program, called GETKEY, could be developed that first reads a sequential file and extracts a specific field defined by the operator. For example, the command

**GETKEY NAMES.DAT LASTNAME 10 20**

would cause GETKEY to read the data base file NAMES.DAT and extract the LASTNAME field from each record, starting in position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. This list is called an inverted index in information retrieval parlance.

If the programmer were to rename the program shown above as QUERY and modify it so that it reads a sorted key file into memory, the command line might appear as

**QUERY NAMES.DAT LASTNAME.KEY**

Instead of reading a number, the QUERY program reads an alphanumeric string that is a particular key to find in the NAMES.DAT data base. Because the LASTNAME.KEY list is sorted, one can find a particular entry rapidly by performing a binary search, similar to looking up a name in the telephone book. Starting at both ends of the list, one examines the entry halfway in between and, if not matched, splits either the upper half or the lower half for the next search. You will quickly reach the item you are looking for and find the corresponding record number. You should fetch and display this record at the console, just as was done in the program shown above.

With some more work, you can allow a fixed grouping size that differs from the 128-byte record shown above. This is accomplished by keeping track of the record number and the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions, which compute the set of records that satisfy several relationships, such as a LAST-NAME between HARDY and LAUREL and an AGE lower than 45. Display all the records that fit this description. Finally, if your lists are getting too big to fit into memory, randomly access key files from the disk as well.

## 5.6 System Function Summary

Function Number	Function Name	Input	Output
Decimal	Hex		
0	0 System Reset	C = 00H	none
1	1 Console Input	C = 01H	A = ASCII char
2	2 Console Output	E = char	none
3	3 Reader Input		A = ASCII char
4	4 Punch Output	E = char	none
5	5 List Output	E = char	none
6	6 Direct Console I/O	C = 06H	A = char or status
		E = OFFH (input) or (no value) OFEH (status) or char (output)	
7	7 Get I/O Byte	none	A = I/O byte Value

8	B	Set I/O Byte	E = I/O Byte	none
9	9	Print String	DE = Buffer Address	none
10	A	Read Console Buffer	DE = Buffer	Console Characters in Buffer
11	B	Get Console Status	none	A = 00/non zero
12	C	Return Version Number	none	HL: Version Number
13	D	Reset Disk System	none	none
14	E	Select Disk	E = Disk Number	none
15	F	Open File	DE = FCB Address	FF if not found
16	10	Close File	DE = FCB Address	FF if not found
17	11	Search For First	DE = FCB Address	A = Directory Code
18	12	Search For Next	none	A = Directory Code
19	13	Delete File	DE = FCB Address	A = none
20	14	Read Sequential	DE = FCB Address	A = Error Code
21	15	Write Sequential	DE = FCB Address	A = Error Code
22	16	Make File	DE = FCB Address	A = FF if no DIR Space
23	17	Rename File	DE = FCB Address	A = FF in not found
24	18	Return Login Vector	none	HL = Login Vector*
25	19	Return Current Disk	none	A = Current Disk Number
26	1A	Set DMA Address	DE = DMA Address	none
27	1B	Get ADDR (ALLOC)	none	HL = ALLOC Address*
28	1C	Write Protect Disk	none	none
29	1D	Get Read/only Vector	none	HL = R/O Vector Value*
30	1E	Set File Attributes	DE = FCB Address	A = none
31	1F	Get ADDR (Disk Params)	none	HL = DPB Address
32	20	Set/Get User Code	E = OFFH for Get E = 00 to OFH for Set	User Number
33	21	Read Random	DE = FCB Address	A = Error Code
34	22	Write Random	DE = FCB Address	A = Error Code
35	23	Compute File Size	DE = FCB Address	r0, r1, r2
36	24	Set Random Record	DE = FCB Address	r0, r1, r2

37	25	Reset Drive	DE = Drive Vector	A = 0
38	26	Access Drive	not supported	
39	27	Free Drive	not supported	
40	28	Write Random with Fill	DE = FCB	A = Error Code

\*Note that A=L, and B=H upon return.

*End of Section 5*

# Section 6

## CP/M 2 Alteration

### 6.1 Introduction

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so you can alter a specific set of subroutines that define the hardware operating environment.

Although standard CP/M 2 is configured for single-density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single-drive minidisks to high-capacity, hard disk systems. To simplify the following adaptation process, it is assumed that CP/M 2 is first configured for single-density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, you might want to review the system generation process and skip to later sections that discuss system alteration for nonstandard disk systems.

To achieve device independence, CP/M is separated into three distinct modules:

- BIOS is the Basic I/O System, which is environment dependent.
- BDOS is the Basic Disk Operating System, which is not dependent upon the hardware configuration.
- CCP is the Console Command Processor, which uses the BDOS.

Of these modules, only the BIOS is dependent upon the particular hardware. You can patch the distribution version of CP/M to provide a new BIOS that provides a customized interface between the remaining CP/M modules and the hardware system. This document provides a step-by-step procedure for patching a new BIOS into CP/M.

All disk-dependent portions of CP/M 2 are placed into a BIOS, a resident disk parameter block, which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2 operation. Deblocking information is provided, which aids in assembly or disassembly of sector sizes that are multiples of the

fundamental 128-byte data unit, and the system alteration manual includes general purpose subroutines that use the deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table-drive data access algorithms, makes CP/M 2 a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2 is structured, however, so that as much as 128K bytes of data are addressed by a single physical extent, corresponding to a single directory entry, maintaining compatibility with previous versions while taking advantage of directory space.

If CP/M is being tailored to a computer system for the first time, the new BIOS requires some simple software development and testing. The standard BIOS is listed in Appendix A and can be used as a model for the customized package. A skeletal version of the BIOS given in Appendix B can serve as the basis for a modified BIOS.

In addition to the BIOS, you must write a simple memory loader, called GETSYS, which brings the operating system into memory. To patch the new BIOS into CP/M, you must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the disk. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 6.4 and listed in Appendix C.

To make the CP/M system load automatically, you must also supply a cold start loader, similar to the one provided with CP/M, listed in Appendixes A and D. A skeletal form of a cold start loader is given in Appendix E, which serves as a model for the loader.

## 6.2 First-level System Regeneration

The procedure to patch the CP/M system is given below. Address references in each step are shown with H denoting the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, a bias is added to each address that is shown with a + b following it, where b is equal to the memory size-20K. Values for b in various standard memory sizes are listed in Table 6-1.

Table 6-1. Standard Memory Size Values

Memory Size	Value
24K:	b = 24K - 20K = 4K = 1000H
32K:	b = 32K - 20K = 12K = 3000H
40K:	b = 40K - 20K = 20K = 5000H
48K:	b = 48K - 20K = 28K = 7000H
56K:	b = 56K - 20K = 36K = 9000H
62K:	b = 62K - 20K = 42K = A800H
64K:	b = 64K - 20K = 44K = B000H

Note that the standard distribution version of CP/M is set for operation within a 20K CP/M system. Therefore, you must first bring up the 20K CP/M system, then configure it for actual memory size (see Section 6.3).

Follow these steps to patch your CP/M system:

1. Read Section 6.4 and write a GETSYS program that reads the first two tracks of a disk into memory. The program from the disk must be loaded starting at location 3380H. GETSYS is coded to start at location 100H (base of the TPA) as shown in Appendix C.
2. Test the GETSYS program by reading a blank disk into memory, and check to see that the data has been read properly and that the disk has not been altered in any way by the GETSYS program.

3. Run the GETSYS program using an initialized CP/M disk to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).
4. Read Section 6.4 and write the PUTSYS program. This writes memory starting at 3380H back onto the first two tracks of the disk. The PUTSYS program should be located at 200H, as shown in Appendix C.
5. Test the PUTSYS program using a blank, uninitialized disk by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, because this program will be used to alter CP/M on disk.
6. Study Sections 6.5, 6.6, and 6.7 along with the distribution version of the BIOS given in Appendix A and write a simple version that performs a similar function for the customized environment. Use the program given in Appendix B as a model. Call this new BIOS by name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive and simple console input/output functions in this phase.
7. Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be careful to ensure that no disk write operations occur during read operations and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks might cause destruction of the initialized CP/M system after it is patched.
8. Referring to Table 6-3 in Section 6.5, note that the BIOS is placed between locations 4A00H and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the CBIOS developed in step 6 and tested in step 7. This replacement is done in memory.
9. Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank disk for testing.
10. Use GETSYS to bring the copied memory image from the test disk back into memory at 3380H and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine initializes page zero, then jumps to the CCP at location 3400H, which calls the BDOS, which calls the CBIOS. The CCP asks the CBIOS to read sixteen sectors on track 2, and CP/M types A>, the system prompt.

If difficulties are encountered, use whatever debug facilities are available to trace and breakpoint the CBIOS.

11. Upon completion of step 10, CP/M has prompted the console for a command input. To test the disk write operation, type

```
SAVE 1 X.COM
```

All commands must be followed by a carriage return. CP/M responds with another prompt after several disk accesses:

```
A>
```

If it does not, debug the disk write functions and retry.

12. Test the directory command by typing

```
DIR
```

CP/M responds with

```
A:X      COM
```

13. Test the erase command by typing

```
ERA X.COM
```

CP/M responds with the A prompt. This is now an operational system that only requires a bootstrap loader to function completely.

14. Write a bootstrap loader that is similar to GETSYS and place it on track 0, sector 1, using PUTSYS (again using the test disk, not the distribution disk). See Sections 6.5 and 6.8 for more information on the bootstrap operation.

15. Retest the new test disk with the bootstrap loader installed by executing steps 11, 12, and 13. Upon completion of these tests, type a CTRL-C. The system executes a warm start, which reboots the system, and types the A prompt.

16. At this point, there is probably a good version of the customized CP/M system on the test disk. Use GETSYS to load CP/M from the test disk. Remove the test disk, place the distribution disk, or a legal copy, into the drive, and use PUTSYS to replace the distribution version with the customized version. Do not make this replacement if you are unsure of the patch because this step destroys the system that was obtained from Digital Research.

17. Load the modified CP/M system and test it by typing

```
DIR
```

CP/M responds with a list of files that are provided on the initialized disk. The file DDT.COM is the memory image for the debugger. Note that from now on, you must always reboot the CP/M system (CTRL-C is sufficient) when the disk is removed and replaced by another disk, unless the new disk is to be Read-Only.

18. Load and test the debugger by typing

DDT

See Section 4 for operating procedures.

19. Before making further CBIOS modifications, practice using the editor (see Section 2), and assembler (see Section 3). Recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program that does a sector-to-sector copy from one disk to another to obtain back-up copies of the original disk. Read the CP/M Licensing Agreement specifying legal responsibilities when copying the CP/M system. Place the following copyright notice:

Copyright (c), 1983  
Digital Research

on each copy that is made with the COPY program.

20. Modify the CBIOS to include the extra functions for punches, readers, and sign-on messages, and add the facilities for additional disk drives, if desired. These changes can be made with the GETSYS and PUTSYS programs or by referring to the regeneration process in Section 6.3.

You should now have a good copy of the customized CP/M system. Although the CBIOS portion of CP/M belongs to the user, the modified version cannot be legally copied.

It should be noted that the system remains file-compatible with all other CP/M systems (assuming media compatibility) which allows transfer of nonproprietary software between CP/M users.

### 6.3 Second-level System Generation

Once the system is running, the next step is to configure CP/M for the desired memory size. Usually, a memory image is first produced with the MOVCPM program (system relocator) and then placed into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger and the system generation program (refer to Section 1).

The CBIOS and BOOT are modified using ED and assembled using ASM, producing files called CBIOS.HEX and BOOT.HEX, which contain the code for CBIOS and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, type the command:

**MOVCPM xx\***

where xx is the memory size in decimal K bytes, for example, 32 for 32K. The response is as follows:

**CONSTRUCTING xxK CP/M VERS 2.0**

**READY FOR "SYSGEN" OR**

**"SAVE 34 CPMxx.COM"**

An image of CP/M in the TPA is configured for the requested memory size. The memory image is at location 0900H through 227FH, that is, the BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H. Note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that you can patch the CBIOS and CBOOT into it:

**SAVE 34 CPMxx.COM**

The memory image created by the MOVCPM program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

**DDT CPMxx.COM**      Loads DDT, then reads the CP/M image.

DDT should respond with the following:

**NEXT PC  
2300 0100**

(The DDT prompt)

You can then give the display and disassembly commands to examine portions of the memory image between 900H and 227FH. Note, however, that to find any particular address within the memory image, you must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01, is loaded to location 900H (the

user should find the cold start loader at 900H to 97FH); track 00, sector 02, is loaded into 980H (this is the base of the CCP); and so on through the entire CP/M system load. In a 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

$$3400H + n = 980H, \text{ or } n = 980H - 3400H$$

Assuming two's complement arithmetic, n = D580H, which can be checked by

$$3400H + D580H = 10980H = 0980H \text{ (ignoring high-order overflow).}$$

Note that for larger systems, n satisfies

$$\begin{aligned} (3400H + b) + n &= 980H, \text{ or} \\ n &= 980H - (3400H + b), \text{ or} \\ n &= D580H - b \end{aligned}$$

The value of n for common CP/M systems is given below.

Table 6-2. Common Values for CP/M Systems

Memory Size	BIAS b	Negative Offset n
20K	0000H	D580H - 0000H = D580H
24K	1000H	D580H - 1000H = C580H
32K	3000H	D580H - 3000H = A580H
40K	5000H	D580H - 5000H = 8580H
48K	7000H	D580H - 7000H = 6580H
56K	9000H	D580H - 9000H = 4580H
62K	A800H	D580H - A800H = 2D80H
64K	B000H	D580H - B000H = 2580H

If you want to locate the address x within the memory image loaded under DDT in a 20K system, first type

Hx ,n                    Hexadecimal sum and difference

and DDT responds with the value of  $x + n$  (sum) and  $x - n$  (difference). The first number printed by DDT is the actual memory address in the image where the data or code is located. For example, the following DDT command:

H3400 ,D580

produces 980H as the sum, which is where the CCP is located in the memory image under DDT.

Type the L command to disassemble portions of the BIOS located at  $(4A00H + b) - n$ , which, when one uses the H command, produces an actual address of 1F80H. The disassembly command would thus be as follows:

L1F80

It is now necessary to patch in the CBOOT and CBIOS routines. The BOOT resides at location 0900H in the memory image. If the actual load address is n, then to calculate the bias (m), type the command:

H900 ,n                    Subtract load address from target address.

The second number typed by DDT in response to the command is the desired bias (m). For example, if the BOOT executes at 0080H, the command

H900 ,80

produces

0980 0880                Sum and difference in hex.

Therefore, the bias m would be 0880H. To read-in the BOOT, give the command:

I CBOOT. HEX              Input file CBOOT.HEX

Then

Rm                        Read CBOOT with a bias of m (= 900H - n).

Examine the CBOOT with

L900

You are now ready to replace the CBIOS by examining the area at 1F80H, where the original version of the CBIOS resides, and then typing

ICBIOS.HEX      Ready the hex file for loading.

Assume that the CBIOS is being integrated into a 20K CP/M system and thus originates at location 4A00H. To locate the CBIOS properly in the memory image under DDT, you must apply the negative bias n for a 20K system when loading the hex file. This is accomplished by typing

RD580      Read the file with bias D580H.

Upon completion of the read, reexamine the area where the CBIOS has been loaded (use an L1F80 command) to ensure that it is properly loaded. When you are satisfied that the change has been made, return from DDT using a CTRL-C or, GO command.

SYSGEN is used to replace the patched memory image back onto a disk (you use a test disk until sure of the patch) as shown in the following interaction:

SYSGEN      Start the SYSGEN program.

SYSGEN VERSION 2.0      Sign-on message from SYSGEN.

SOURCE DRIVE NAME  
(OR RETURN TO SKIP)      Respond with a carriage return to skip the CP/M read operation because the system is already in memory.

DESTINATION DRIVE NAME  
(OR RETURN TO REBOOT)      Respond with B to write the new system to the disk in drive B.

DESTINATION ON B,  
THEN TYPE RETURN      Place a scratch disk in drive B, then press RETURN.

FUNCTION COMPLETE  
DESTINATION DRIVE NAME  
(OR RETURN TO REBOOT)

Place the scratch disk in drive A, then perform a cold start to bring up the newly-configured CP/M system.

The new CP/M system is then tested and the Digital Research copyright notice is placed on the disk, as specified in the Licensing Agreement:

Copyright (c), 1979  
Digital Research

## 6.4 Sample GETSYS and PUTSYS Programs

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Sections 6.1 and 6.2. To read and write the specific sectors, you must insert the READSEC and WRITESEC subroutines.

```
; GETSYS PROGRAM -- READ TRACKS 0 AND 1 TO MEMORY AT 3380H
; REGISTER           USE
;
;      A             (SCRATCH REGISTER)
;
;      B             TRACK COUNT (0, 1)
;
;      C             SECTOR COUNT (1,2,...,26)
;
;      DE            (SCRATCH REGISTER PAIR)
;
;      HL            LOAD ADDRESS
;
;      SP            SET TO STACK ADDRESS
;
;
START: LXI SP,3380H    ;SET STACK POINTER TO SCRATCH
       ;AREA
       LXI H,3380H    ;SET BASE LOAD ADDRESS
       MVI B,0        ;START WITH TRACK 0
RDTRK: MVI C,1        ;READ NEXT TRACK (INITIALLY 0)
       ;READ STARTING WITH SECTOR 1
```

```

RDSEC:           ;READ NEXT SECTOR
    CALL READSEC ;USER-SUPPLIED SUBROUTINE
    LXI D,128   ;MOVE LOAD ADDRESS TO NEXT 1/2
                  ;PAGE
    DAD D       ;HL = HL + 128
    INR C       ;SECTOR = SECTOR + 1
    MOV A,C     ;CHECK FOR END OF TRACK
    CPI 27
    JC RDSEC   ;CARRY GENERATED IF SECTOR <27

;
; ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
    INR B
    MOV A,B     ;TEST FOR LAST TRACK
    CPI 2
    JC RDTRK   ;CARRY GENERATED IF TRACK <2

;
; USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
; ENTER WITH TRACK NUMBER IN REGISTER B,
; SECTOR NUMBER IN REGISTER C, AND
;
; ADDRESS TO FILL IN HL
;
; PUSH B      ;SAVE B AND C REGISTERS
; PUSH H      ;SAVE HL REGISTERS

.....
; Perform disk read at this point, branch to
; label START if an error occurs
.....
; POP H      ;RECOVER HL
; POP B      ;RECOVER B AND C REGISTERS
; RET        ;BACK TO MAIN PROGRAM

END START

```

Listing 6-1. GETSYS Program

This program is assembled and listed in Appendix B for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes that are listed on the left might be useful if the program has to be entered through the panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix C. The register pair HL becomes the dump address, next address to write, and operations on these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine, which performs the opposite function; data from address HL is written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in Appendix C.

## 6.5 Disk Organization

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector contains an optional software boot section (see the table on the following page). Disk controllers are often set up to bring track 0, sector 1, into memory at a specific location, often location 0000H. The program in this sector, called BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 3400H + b. If the controller does not have a built-in sector load, the program in track 0, sector 1 can be ignored. In this case, load the program from track 0, sector 2, to location 3400H + b.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1, into absolute address 3000H. Upon loading this sector, control transfers to location 3000H, where the bootstrap operation commences by loading the remainder of track 0 and all of track 1 into memory, starting at 3400H + b. Note that this bootstrap loader is of little use in a non-MDS environment, although it is useful to examine it because some of the boot actions will have to be duplicated in the user's cold start loader.

Table 6-3. CP/M Disk Sector Allocation

<i>Track #</i>	<i>Sector</i>	<i>Page #</i>	<i>Memory Address</i>	<i>CP/M Module name</i>
00	01		(boot address)	Cold Start Loader
00	02	00	3400H +b	CCP
,	03	,	3480H +b	,
,	04	01	3500H +b	,
,	05	,	3580H +b	,
,	06	02	3600H +b	,
,	07	,	3680H +b	,
,	08	03	3700H +b	,
,	09	,	3780H +b	,
,	10	04	3800H +b	,
,	11	,	3880H +b	,
,	12	05	3900H +b	,
,	13	,	3980H +b	,
,	14	06	3A00H +b	,
,	15	,	3A80H +b	,
,	16	07	3B00H +b	,
00	17	,	3B80H +b	CCP
00	18	08	3C00H +b	BDOS
,	19	,	3C80H +b	,
,	20	09	3D00H +b	,
,	21	,	3D80H +b	,
,	22	10	3E00H +b	,
,	23	,	3E80H +b	,
,	24	11	3F00H +b	,
,	25	,	3F80H +b	,
,	26	12	4000H +b	,
01	01	,	4080H +b	,
,	02	13	4100H +b	,
,	03	,	4180H +b	,
,	04	14	4200H +b	,
,	05	,	4280H +b	,
,	06	15	4300H +b	,
,	07	,	4380H +b	,
,	08	16	4400H +b	,
,	09	,	4480H +b	,
,	10	17	4500H +b	,
,	11	,	4580H +b	,

Table 6-3. (continued)

<i>Track #</i>	<i>Sector</i>	<i>Page #</i>	<i>Memory Address</i>	<i>CP/M Module name</i>
,	12	18	4600H + b	,
,	13	,	4680H + b	,
,	14	19	4700H + b	,
,	15	,	4780H + b	,
,	16	20	4800H + b	,
,	17	,	4880H + b	,
,	18	21	4900H + b	,
01	19	,	4900H + b	BDOS
07	20	22	4A00H + b	BIOS
,	21	,	4A80H + b	,
,	22	23	4B00H + b	,
,	23	,	4B80H + b	,
,	24	24	4C00H + b	,
01	25	,	4C80H + b	BIOS
01	26	25	4D00H + b	BIOS
02-76	01-26			(directory and data)

## 6.6 The BIOS Entry Points

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a jump vector located at 4A00H + b, as shown below. See Appendixes A and B. The jump vector is a sequence of 17 jump instructions that send program control to the individual BIOS subroutines. The BIOS subroutines might be empty for certain functions (they might contain a single RET operation) during reconfiguration of CP/M, but the entries must be present in the jump vector.

The jump vector at 4A00H + b takes the form shown below, where the individual jump addresses are given to the left:

4A00H+b	JMP BOOT	;ARRIVE HERE FROM COLD START LOAD
4A03H+b	JMP WBOOT	;ARRIVE HERE FOR WARM START
4A06H+b	JMP CONST	;CHECK FOR CONSOLE CHAR READY

4A09H+b	JMP CONIN	;READ CONSOLE CHARACTER IN
4A0CH+b	JMP CONOUT	;WRITE CONSOLE CHARACTER OUT
4A0FH+b	JMP LIST	;WRITE LISTING CHARACTER OUT
4A12H+b	JMP PUNCH	;WRITE CHARACTER TO PUNCH DEVICE
4A15H+b	JMP READER	;READ READER DEVICE
4A18H+b	JMP HOME	;MOVE TO TRACK 00 ON SELECTED DISK
4A1BH+b	JMP SELDSK	;SELECT DISK DRIVE
4A1EH+b	JMP SETTRK	;SET TRACK NUMBER
4A21H+b	JMP SETSEC	;SET SECTOR NUMBER
4A24H+b	JMP SETDMA	;SET DMA ADDRESS
4A27H+b	JMP READ	;READ SELECTED SECTOR
4A2AH+b	JMP WRITE	;WRITE SELECTED SECTOR
4A2DH+b	JMP LISTST	;RETURN LIST STATUS
4A30H+b	JMP SECTRAN	;SECTOR TRANSLATE SUBROUTINE

Listing 6-2. BIOS Entry Points

Each jump address corresponds to a particular subroutine that performs the specific function, as outlined below. There are three major divisions in the jump table: the system reinitialization, which results from calls on BOOT and WBOOT; simple character I/O, performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST; and disk I/O, performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper- and lower-case, with high-order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII CTRL-Z (1AH). Peripheral devices are seen by CP/M as logical devices and are assigned to physical devices within the BIOS.

To operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines. LIST, PUNCH, and READER can be used by PIP, but not the BDOS. Further, the LISTST entry is currently used only by DESPOOL, the print spooling utility. Thus, the initial version of CBIOS can have empty subroutines for the remaining ASCII devices.

The following list describes the characteristics of each device.

- CONSOLE is the principal interactive console that communicates with the operator and it is accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or teletype.
- LIST is the principal listing device. If it exists on the user's system, it is usually a hard-copy device, such as a printer or teletype.
- PUNCH is the principal tape punching device. If it exists, it is normally a high-speed paper tape punch or teletype.
- READER is the principal tape reading device, such as a simple optical reader or teletype.

A single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS gives an appropriate error message so that the system does not hang if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a 1AH (CTRL-Z) in register A to indicate immediate end-of-file.

For added flexibility, you can optionally implement the IOBYTE function, which allows reassignment of physical devices. The IOBYTE function creates a mapping of logical-to-physical devices that can be altered during CP/M processing, see the STAT command in Section 1.6.1.

The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory, currently location 0003H, is maintained, called IOBYTE, which defines the logical-to-physical device mapping that is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown in the following figure.

	MOST SIGNIFICANT		LEAST SIGNIFICANT	
IOBYTE AT 003H	LIST	PUNCH	READER	CONSOLE
	BITS 6, 7	BITS 4, 5	BITS 2, 3	BITS 0, 1

Figure 6-1. IOBYTE Fields

The value in each field can be in the range 0–3, defining the assigned source or destination of each logical device. Table 6-4 gives the values that can be assigned to each field.

Table 6-4. IOBYTE Field Values

Value	Meaning
CONSOLE field (bits 0,1)	
0	console is assigned to the console printer device (TTY:)
1	console is assigned to the CRT device (CRT:)
2	batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
3	user-defined console device (UC1:)
READER field (bits 2,3)	
0	READER is the teletype device (TTY:)
1	READER is the high speed reader device (PTR:)
2	user-defined reader #1 (UR1:)
3	user-defined reader #2 (UR2:)

Table 6-4. (continued)

Value	Meaning
PUNCH field (bits 4,5)	
0	PUNCH is the teletype device (TTY:)
1	PUNCH is the high speed punch device (PTP:)
2	user-defined punch #1 (UP1:)
3	user-defined punch #2 (UP2:)
LIST field (bits 6,7)	
0	LIST is the teletype device (TTY:)
1	LIST is the CRT device (CRT:)
2	LIST is the line printer device (LPT:)
3	user-defined list device (UL1:)

The implementation of the IOBYTE is optional and effects only the organization of the CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H) except for PIP, which allows access to the physical devices, and STAT, which allows logical-physical assignments to be made or displayed. For more information see Section 1. In any case the IOBYTE implementation should be omitted until the basic CBIOS is fully implemented and tested; then you should add the IOBYTE to increase the facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines that set up the disk number to access, the track and sector on a particular disk, and the Direct Memory Access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation.

There is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there might be a single call to set the DMA address, followed by several calls that read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it reports the error to the user. The HOME subroutine might or might not actually perform the track 00 seek, depending upon controller characteristics; the important point is that track 00 has been selected for the next operation and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The following table describes the exact responsibilities of each BIOS entry point subroutine.

Table 6-5. BIOS Entry Points

<i>Entry Point</i>	<i>Function</i>
BOOT	The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a sign-on message, which can be omitted in the first version. If the IOBYTE function is implemented, it must be set at this point. The various system parameters that are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400 + b for further processing. Note that register C must be set to zero to select drive A.
WBOOT	The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS, or CBIOS, if the user has completed the patch. System parameters must be initialized as follows:
location 0,1,2	Set to JMP WBOOT for warm starts (000H: JMP 4A03H + b)
location 3	Set initial value of IOBYTE, if implemented in the CBIOS
location 4	High nibble = current user no; low nibble = current drive

Table 6-5. (continued)

<i>Entry Point</i>	<i>Function</i>
	location 5,6,7      Set to JMP BDOS, which is the primary entry point to CP/M for transient programs. (0005H: JMP 3C06H + b)
	Refer to Section 6.9 for complete details of page zero use. Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H + b to restart the system. Upon entry to the CCP, register C is set to the drive to select after system initialization. The WBOOT routine should read location 4 in memory, verify that is a legal drive, and pass it to the CCP in register C.
CONST	You should sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.
CONIN	The next console character is read into register A, and the parity bit is set, high-order bit, to zero. If no console character is ready, wait until a character is typed before returning.
CONOUT	The character is sent from register C to the console output device. The character is in ASCII, with high-order parity bit set to zero. You might want to include a time-out on a line-feed or carriage return, if the console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can filter out control characters that cause the console device to react in a strange way (CTRL-Z causes the Lear-Siegler terminal to clear the screen, for example).
LIST	The character is sent from register C to the currently assigned listing device. The character is in ASCII with zero parity bit.
PUNCH	The character is sent from register C to the currently assigned punch device. The character is in ASCII with zero parity.
READER	The next character is read from the currently assigned reader device into register A with zero parity (high-order bit must be zero); an end-of-file condition is reported by returning an ASCII CTRL-Z(1AH).

Table 6-5. (continued)

<i>Entry Point</i>	<i>Function</i>
HOME	The disk head of the currently selected disk (initially disk A) is moved to the track 00 position. If the controller allows access to the track 0 flag from the drive, the head is stepped until the track 0 flag is detected. If the controller does not support this feature, the HOME call is translated into a call to SETTRK with a parameter of 0.
SELDSK	<p>The disk drive given by register C is selected for further operations, where register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in Section 6.10. For standard floppy disk drives, the contents of the header and associated tables do not change; thus, the program segment included in the sample CBIOS performs this operation automatically.</p>
	<p>If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the physical disk select operation until an I/O function (seek, read, or write) is actually performed, because disk selects often occur without ultimately performing any disk I/O, and many controllers unload the head of the current disk before selecting the new drive. This causes an excessive amount of noise and disk wear. The least significant bit of register E is zero if this is the first occurrence of the drive select since the last cold or warm start.</p>
SETTRK	Register BC contains the track number for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. You can choose to seek the selected track at this time or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives and 0-65535 for non-standard disk subsystems.

Table 6-5. (continued)

<i>Entry Point</i>	<i>Function</i>
SETSEC	Register BC contains the sector number, 1 through 26, for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SEC-TRAN entry point. You can choose to send this information to the controller at this point or delay sector selection until a read or write operation occurs.
SETDMA	Register BC contains the DMA (Disk Memory Access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, all subsequent read operations read their data into 80H through OFFH and all subsequent write operations get their data from 80H through OFFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. The controller need not actually support Direct Memory Access. If, for example, all data transfers are through I/O ports, the CBIOS that is constructed uses the 128-byte area starting at the selected DMA address for the memory buffer during the subsequent read or write operations.
READ	Assuming the drive has been selected, the track has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters and returns the following error codes in register A:  0 no errors occurred  1 nonrecoverable error condition occurred  Currently, CP/M responds only to a zero or nonzero value as the return code. That is, if the value in register A is 0, CP/M assumes that the disk operation was completed properly. IF an error occurs the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS prints the message BDOS ERR ONx: BAD SECTOR. The operator then has the option of pressing a carriage return to ignore the error, or CTRL-C to abort.

Table 6-5. (continued)

<i>Entry Point</i>	<i>Function</i>
WRITE	Data is written from the currently selected DMA address to the currently selected drive, track, and sector. For floppy disks, the data should be marked as nondeleted data to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.
LISTST	You return the ready status of the list device used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character and OFFH if a character can be sent to the printer. A 00 value should be returned if LIST status is not implemented.
SECTRAN	<p>Logical-to-physical sector translation is performed to improve the overall response of CP/M. Standard CP/M systems are shipped with a skew factor of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems that use fast processors, memory, and disk subsystems, the skew factor might be changed to improve overall response. However, the user should maintain a single-density IBM-compatible version of CP/M for information transfer into and out of the computer system, using a skew factor of 6.</p> <p>In general, SECTRAN receives a logical sector number relative to zero in BC and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the table and indexing code is provided in the CBIOS and need not be changed.</p>

## 6.7 A Sample BIOS

The program shown in Appendix B can serve as a basis for your first BIOS. The simplest functions are assumed in this BIOS, so that you can enter it through a front panel, if absolutely necessary. You must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see Section 6.9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled out and the IOBYTE function can be implemented.

## 6.8 A Sample Cold Start Loader

The program shown in Appendix E can serve as a basis for a cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Space is reserved for the patch code so that the total amount of storage required for the cold start loader is 128 bytes.

Eventually, you might want to get this loader onto the first disk sector (track 0, sector 1) and cause the controller to load it into memory automatically upon system start up. Alternatively, the cold start loader can be placed into ROM, and above the CP/M system. In this case, it is necessary to originate the program at a higher address and key in a jump instruction at system start up that branches to the loader. Subsequent warm starts do not require this key-in operation, because the entry point WBOOT gets control, thus bringing the system in from disk automatically. The skeletal cold start loader has minimal error recovery, which might be enhanced in later versions.

## 6.9 Reserved Locations in Page Zero

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data that are used during CP/M processing. The code and data areas are given in the following table.

Table 6-6. Reserved Locations in Page Zero

<i>Locations</i>	<i>Contents</i>
000H-0002H	Contains a jump instruction to the warm start entry location 4A03H + b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H-0003H	Contains the Intel standard IOBYTE is optionally included in the user's CBIOS (refer to Section 6.6).
0004H-0004H	Current default drive number (0 = A, . . . , 15 = P).
0005H-0007H	Contains a jump instruction to the BDOS and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in Section 5, and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M, assuming the CCP is being overlaid. The DDT program changes the address field to reflect the reduced memory size in debug mode.
0008H-0027H	Interrupt locations 1 through 5 not used.
0030H-0037H	Interrupt location 6 (not currently used) is reserved.
0038H-003AH	Restart 7; contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH-003FH	Not currently used; reserved.

Table 6-6. (continued)

<i>Locations</i>	<i>Contents</i>
0040H-004FH	A 16-byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M.
0050H-005BH	Not currently used; reserved.
005CH-007CH	Default File Control Block produced for a transient program by the CCP.
007DH-007FH	Optional default random record position.
0080H-00FFH	Default 128-byte disk buffer, also filled with the command line when a transient is loaded under the CCP.

This information is set up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can first be loaded into the TPA, using normal CP/M facilities, with a small memory move program that gets control when loaded. The memory move program must get control from location 0100H, which is the assumed beginning of all transient programs. The move program can then proceed to the entire memory image down to location 0 and pass control to the starting address of the memory load.

If the BIOS is overwritten or if location 0, containing the warm start entry point, is overwritten, the operator must bring the CP/M system back into memory with a cold start sequence.

## 6.10 Disk Parameter Tables

Tables are included in the BIOS that describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix B, or automatically generated using the DISKDEF macro library, as shown in Appendix F. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header that contains information about the disk drive and provides a scratch pad area for certain BDOS operations. The format of the disk parameter header for each drive is shown in Figure 6-2, where each element is a word (16-bit) value.

<b>DISK PARAMETER HEADER</b>							
XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16B	16B	16B	16B	16B	16B	16B	16B

Figure 6-2. Disk Parameter Header Format

The meaning of each Disk Parameter Header (DPH) element is detailed in Table 6-7.

Table 6-7. Disk Parameter Headers

<i>Disk Parameter Header</i>	<i>Meaning</i>
XLT	Address of the logical-to-physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (that is, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratch pad values for use within the BDOS, initial value is unimportant.
DIRBUF	Address of a 128-byte scratch pad area for directory operations within BDOS. All DPHs address the same scratch pad area.

Table 6-7. (continued)

Disk Parameter Header	Meaning
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Address of a scratch pad area used for software check for changed disks. This address is different for each DPH.
ALV	Address of a scratch pad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPHs are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. In the following figure the label DPBASE defines the base address of the DPH table.

DPBASE:

00	XLT 00	0000	0000	0000	DIRBUF	DBP 00	CSV 00	ALV 00
01	XLT 01	0000	0000	0000	DIRBUF	DBP 01	CSV 01	ALV 01
(AND SO ON THROUGH)								
n-1	XLTn-1	0000	0000	0000	DIRBUF	DBPn-1	CSVn-1	ALVn-1

Figure 6-3. Disk Parameter Header Table

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```

NDISK$      EQU     4          ;NUMBER OF DISK DRIVES
.....
SELDSK:    ;SELECT DISK GIVEN BY BC
    LSI      H,0000H    ;ERROR CODE
    MOV      A,C        ;DRIVE OK?
    CPI      NDISKS    ;CY IF SO
    RNC      ;RET IF ERROR
    ;NO ERROR, CONTINUE
    MOV      L,C        ;LOW(DISK)
    MOV      H,B        ;HIGH(DISK)
    DAD      H          ;*2
    DAD      H          ;*4
    DAD      H          ;*8
    DAD      H          ;*16
    LXI      D,DPBASE;FIRST DPH
    DAD      D          ;DPH(DISK)
    RET

```

The translation vectors, XLT 00 through XLT<sub>n-1</sub>, are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count 1. The Disk Parameter Block (DPB) for each drive is more complex. As shown in Figure 6-4, particular DPB, that is addressed by one or more DPHs, takes the general form:

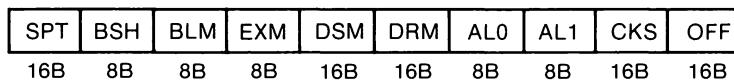


Figure 6-4. Disk Parameter Block Format

where each is a byte or word value, as shown by the 8b or 16b indicator below the field.

The following field abbreviations are used in Figure 6-4:

- SPT is the total number of sectors per track.
- BSH is the data allocation block shift factor, determined by the data block allocation size.
- BLM is the data allocation block mask ( $2^{[BSH-1]}$ ).
- EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.
- DSM determines the total storage capacity of the disk drive.
- DRM determines the total number of directory entries that can be stored on this drive. AL0, AL1 determine reserved directory blocks.
- CKS is the size of the directory check vector.
- OFF is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine the data allocation size BLS, which is not an entry in the DPB. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in Table 6-8.

Table 6-8. BSH and BLM Values

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in Table 6-9.

Table 6-9. EXM Values

BLS	EXM values	
	DSM <256	DSM>255
1,024	0	N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product (DSM + 1) is the total number of bytes held by the drive and must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries that can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The values AL0 and AL1 can together be considered a string of 16-bits, as shown in Figure 6-5.

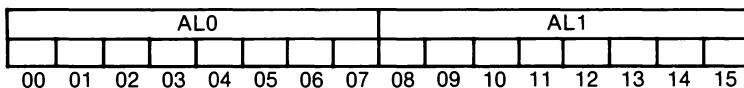


Figure 6-5. AL0 and AL1

Position 00 corresponds to the high-order bit of the byte labeled AL0 and 15 corresponds to the low-order bit of the byte labeled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the following tabulation:

**Table 6-10. BLS Tabulation**

BLS	Directory Entries
1,024	32 times # bits
2,048	64 times # bits
4,096	128 times # bits
8,192	256 times # bits
16,384	512 times # bits

Thus, if DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high-order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM + 1)/4, where DRM is the last directory entry number. If the media are fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called and can be used as a mechanism for skipping reserved operating system tracks or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, several DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH; because the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to DPH for a particular drive, the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If  $CKS = (DRM + 1)/4$ , you must reserve  $(DRM + 1)/4$  bytes for directory check use. If  $CKS = 0$ , no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk and is computed as  $(DSM/8) + 1$ .

The CBIOS shown in Appendix B demonstrates an instance of these tables for standard 8-inch, single-density drives. It might be useful to examine this program and compare the tabular values with the definitions given above.

## 6.11 The DISKDEF Macro Library

A macro library called DISKDEF (shown in Appendix F), greatly simplifies the table construction process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2 distribution disks.

A BIOS disk definition consists of the following sequence of macro statements:

MACLIB	DISKDEF
.....	
DISKS	n
DISKDEF	0,..
DISKDEF	1,..
.....	
DISKDEF	n - 1
.....	
ENDEF	

where the MACLIB statement loads the DISKDEF.LIB file, on the same disk as the BIOS, into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with the user's system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow that define the characteristics of each logical disk, 0 through n - 1, corresponding to logical drives A through P. The DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section and thus must be placed in a nonexecutable portion of the BIOS, typically directly following the BIOS jump vector.

The remaining portion of the BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas that are located in memory above the BIOS.

The DISKDEF macro call takes the form:

```
DISKDEF dn,fsc,lsc,[skf],bls dks,dir,cks,ofs,[0]
```

where

- dn is the logical disk number, 0 to n – 1.
- fsc is the first physical sector number (0 or 1).
- lsc is the last sector number.
- skf is the optional sector skew factor.
- bls is the data allocation block size.
- dks is the number of blocks on the disk.
- dir is the number of directory entries.
- cks is the number of checked directory entries.
- ofs is the track offset to logical track 00.
- [0] is an optional 1.4 compatibility flag.

The value dn is the drive number being defined with this DISKDEF macro invocation. The fsc parameter accounts for differing sector numbering systems and is usually 0 to 1. The lsc is the last numbered sector on a track. When present, the skf parameter defines the sector skew factor, which is used to create a sector translation table according to the skew.

If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted, or equal to 0.

The bls parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes because there are fewer directory references, and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident RAM space is reduced.

The dks parameter specifies the total disk size in bls units. That is, if the bls = 2048 and dks = 1000, the total disk capacity is 2,048,000 bytes. If dks is greater than 255, the block size parameter bls must be greater than 1024. The value of dir is the total number of directory entries that might exceed 255, if desired.

The cks parameter determines the number of directory items to check on each directory scan and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred. When this situation is detected, CP/M automatically marks the disk Read-Only so that data is not subsequently destroyed.

As stated in the previous section, the value of cks = dir when the medium is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, the value of cks is typically 0, because the probability of changing disks without a restart is low.

The ofs value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 that have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form:

DISKDEF i,j

gives disk i the same characteristics as a previously defined drive j. A standard four-drive, single-density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS      4
DISKDEF    0,1,26,6,1024,243,64,2
DISKDEF    1,0
DISKDEF    2,0
DISKDEF    3,0
.....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track, numbered 1 through 26, with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243K-byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n DPHs, starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four-drive standard system, for example, the DISKS macro generates a table of the form:

DPBASE	EQU\$
DPE0:	DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1:	DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2:	DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3:	DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the DPH are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the RAM area following the BIOS code and tables.

Note that if the skf (skew factor) parameter is omitted, or equal to 0, the translation table is omitted and a 0000H value is inserted in the XLT position of the DPH for the disk. In a subsequent call to perform the logical-to-physical translation, SECTRAN receives a translation table address of DE = 0000H and simply returns the original logical sector from BC in the HL register pair.

A translate table is constructed when the skf parameter is present, and the (nonzero) table address is placed into the corresponding DPHs. The following, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

XLT0:	DB 1,7,13,19,25,5,11,17,23,3,9,15,21
	DB 2,8,14,20,26,6,12,18,24,4,10,16,22

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS that is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce the following EQU statement:

```
4C72 =      BEGDAT EQU $  
             (data areas)  
  
4DB0 =      ENDDAT EQU $  
  
013C =      DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check drive characteristics, because STAT uses the disk parameter block to decode the drive information. A STAT command of the form:

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A, . . . , P) and displays the following values:

- r: 128-byte record capacity
- k: kilobyte drive capacity
- d: 32-byte directory entries
- c: checked directory entries
- e: records/extent
- b: records/block
- s: sectors/track
- t: reserved tracks

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values. The last example produces a full 8-megabyte system.

DISKDEF 0,1,58,,2048,256,128,128,2  
r = 4096, k = 512, d = 128, c = 128, e = 256, b = 16, s = 58, t = 2

DISKDEF 0,1,58,,2048,1024,300,0,2  
r = 16348, k = 2048, d = 300, c = 0, e = 128, b = 16, s = 58, t = 2

DISKDEF 0,1,58,,16348,512,128,128,2  
r = 65536, k = 8192, d = 128, c = 128, e = 1024, b = 128, s = 58, t = 2

## 6.12 Sector Blocking and Deblocking

Upon each call to BIOS WRITE entry point, the CP/M BDOS includes information that allows effective sector blocking and deblocking where the host disk subsystem has a sector size that is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm that can be included within the BIOS and that uses the BDOS information to perform the operations automatically.

On each call to WRITE, the BDOS provides the following information in register C:

- 0 = (normal sector write)
- 1 = (write to directory sector)
- 2 = (write to the first sector  
of a new data block)

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update; when the write is to other than the first sector of an unallocated block; or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence; thus, there is little overhead involved in either operation when blocking and deblocking records, because preread operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form; this file is included on your CP/M disk. Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer that is the size of the host disk sector. Throughout the program, values and variables that relate to the CP/M sector involved in a seek operation are prefixed by sek, while those related to the host disk system are prefixed by hst. The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK and SETMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hstrk is the host track number, and hstsec is the host sector number, which may require translation to physical sector number. You must insert code at this point that performs the full sector read or write into or out of the buffer at hstbuf of length htsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80-megabyte hard disk unit that was originally configured for 128-byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512-byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain 128-byte sectors. This is primarily because of the information provided by the BDOS, which eliminates the necessity for preread operations.

*End of Section 6*

# Appendix A

## The MDS Basic Input/Output System (BIOS)

This listing has been printed broadside on the page, so that you can easily read the entire listing.

```
1    mds-800 i/o drivers for cpm 2.2
2    (four drive single density version)
3
4    version 2.2 february, 1980
5
6    0016 = vers      equ 22        ;version 2.2
7
8    0018 = bios      equ 03400h   ;base of cop in test system
9    0019 = if         test
10   001a =         0ffh       off
11   001b =         not true
12   001c =         false
13   001d =         true
14   001e =         fffh       value of "true"
15   001f =         0000      = 0000
16   0020 =         0000      =
17
18   0021 =         bios      endif
19
20
```

```

21
22    0000 =      bias      if      not test      igenerate relocatable CP/m system
23
24    1600 =      ;      equ      0000h
25    1600 =      ;      endif
26
27    1600 =      ;      org      1600h
28    0000 =      cPmb      org      Patch
29    0B06 =      bdos      $-Patch      ibase of CPM console Processor
30    1600 =      cPm1      $-Patch      ibasic dos (resident portion)
31    002C =      nsects      806h+cPmb      length (in bytes) of CPM system
32    0002 =      offset      cPm1/128      number of sectors to load
33    0004 =      cdisk      2      number of disk tracks used by CP/m
34    0080 =      buff       0004h      address of last logged disk on warm start
35    000A =      retry      0080h      default buffer address
36
37    ;      equ      10      imax retries on disk i/o before error
38
39
40
41
42
43
44
45
46
47
48
49

```

; Perform following functions

boot      cold start

wboot      warm start (save i/o byte)

(boot and wboot are the same for mds)

const      console status

reg-a = 00 if no character ready

reg-a = ff if character ready

console character in (result in reg-a)

console character out (char in reg-c)

list out (char in reg-c)

punch out (char in reg-c)

paper tape reader in (result to reg-a)

move to track 00

```

50
51   ; (the following calls set-up the io parameter block for the
52   ; mds, which is used to perform subsequent reads and writes)
53   ;      seldisk    select disk given by reg-c (0, 1, 2...)
54   ;      settrk     set track address (0,...,76) for subsequent read-write
55   ;      setsec     set sector address (1,...,26) for subsequent read-write
56   ;      setdma     set subsequent dma.address (initially 80h)
57
58   ; (read and write assume previous calls to set up the io parameters)
59   ;      read       read track/sector to Preset dma address
60   ;      write     write track/sector from Preset dma address
61
62   ;      jump vector for individual routines
63   1600 c3b316
64   1603 c3c316
65   1605 c36117
66   1609 c36417
67   160c c36a17
68   160f c36d17
69   1612 c37217
70   1615 c37517
71   1618 c37817
72   161b c37d17
73   161e c3a717
74   1621 c3ac17
75   1624 c2bb17
76   1627 c3c117
77   162a c3ca17
78   162d c37017
79   1630 c3b117

    ; wboot:          boot
    ; const:          wboot
    ; conin:          const
    ; conout:         conin
    ; list:           conout
    ; punch:          list
    ; reader:         punch
    ; home:           reader
    ; seldsk:         home
    ; settrk:         seldsk
    ; setsec:         settrk
    ; setdma:         setsec
    ; read:           setdma
    ; write:          read
    ; listst:         write
    ; sectran:        listst
    ; iplist status: sectran

```

```

80
81      maclib    diskdef   load the disk definition library
82      disks     4          four disks
83      base      $          base of disk parameter blocks
84      dPbase   equ        xl0, 0000h  translate table
85      dPe0:    dw         0000h, 0000h  iscratch area
86      dPe1:    dw         dirbuf, dpb0
87      dPe2:    dw         csv0, alv0
88      dPe3:    dw         xlt1, 0000h  iscratch area
89      dPe4:    dw         0000h, 0000h  iscratch area
90      dPe5:    dw         dirbuf, dpb1
91      dPe6:    dw         csv1, alv1
92      dPe7:    dw         xlt2, 0000h  iscratch area
93      dPe8:    dw         0000h, 0000h  iscratch area
94      dPe9:    dw         dirbuf, dpb2
95      dPea:    dw         csv2, alv2
96      dPeb:    dw         xlt3, 0000h  iscratch area
97      dPec:    dw         0000h, 0000h  iscratch area
98      dPed:    dw         dirbuf, dpb3
99      dPef:    dw         csv3, alv3
100     diskdef  $          0, 1, 26, 6, 1024, 243, 64, offset
101     dPb0    equ        $          idisk param block
102     dPb1    dw         26        isec per track
103     dPb2    db         3          iblock shift
104     dPb3    db         7          iblock mask
105     dPb4    db         0          iextnt mask
106     dPb5    dw         242       idisk size-1
107     dPb6    dw         63        idirectory max
108     dPb7    db         192       ialloc0
109     dPb8    db         0          ialloc1
110     dPb9    dw         16        icheck size

```

			offset
			translate table
111	1680+0200		2
112	1682+=	xlt0	\$
113	1682+01		1
114	1683+07		7
115	1684+0d	db	13
116	1685+13	db	19
117	1686+19	db	25
118	1687+05	db	5
119	1688+0b	db	11
120	1689+11	db	17
121	168a+17	db	23
122	168b+03	db	3
123	168c+09	db	9
124	168d+0f	db	15
125	168e+15	db	21
126	168f+02	db	2
127	1690+08	db	8
128	1691+0e	db	14
129	1692+14	db	20
130	1693+1a	db	26
131	1694+06	db	6
132	1695+0c	db	12
133	1696+12	db	18
134	1697+18	db	24
135	1698+04	db	4
136	1699+0a	db	10
137	169a+10	db	16
138	169b+16	db	22

```

139      diskdef 1,0
140      dPb1    equ     db0
141      als1    equ     al50
142      css1    equ     c550
143      xl1t1   equ     xl1t0
144      diskdef 2, 0
145      dPb2    equ     dPb0
146      als2    equ     al50
147      css2    equ     c550
148      xl1t2   equ     xl1t0
149      diskdef 3, 0
150      dPb3    equ     dPb0
151      als3    equ     al50
152      css3    equ     c550
153      xl1t3   equ     xl1t0
154      ;       endeef occurs at end of assembly
155      ;
156      ;       end of controller-independent code, the remaining subroutines
157      ;       are tailored to the particular operating environment, and must
158      ;       be altered for any system which differs from the intel mds,
159      ;
160      ;       the following code assumes the mds monitor exists at 0f800h
161      ;       and uses the i/o subroutines within the monitor
162      ;
163      ;       we also assume the mds system has four disk drives
164      revrt   equ     0f4h
165      intc    equ     0fch
166      icon    equ     0f3h
167      inte    equ     0111$1110b

```

© DIGITAL RESEARCH™

```

168
169      ; mds monitor equates
170      mon80    equ     0f800h   ; mds monitor
171      monb0    equ     0ff0fh   ; restart mon80 (boot error)
172      ci       equ     0f803h   ; console character to reg-a
173      ri       equ     0f806h   ; reader in to reg-a
174      co       equ     0f809h   ; console char from C to console out
175      po       equ     0f80ch   ; punch char from C to punch device
176      lo       equ     0f80fh   ; list from C to list device
177      osts    equ     0f812h   ; console status 00/ff to register a
178      ;
179      ; disk Ports and Commands
180      base     equ     78h    ; base of disk command io ports
181      dstat   equ     base    ; disk status (input)
182      rtype   equ     base+1  ; result type (input)
183      rbyte   equ     base+3  ; result byte (input)
184      ;
185      ilow    equ     base+1  ; iopb low address (output)
186      ihigh   equ     base+2  ; iopb high address (output)
187      ;
188      readf   equ     4h     ; read function
189      wrtf    equ     6h     ; write function
190      recal   equ     3h     ; recalibrate drive
191      iondy  equ     4h     ; i/o finished mask
192      cr      equ     0dh    ; carriage return
193      lf      equ     0ah    ; line-feed

```

```

194
195      signon:  ;signon message: XXX CP/M vers yy
196      db      cr, lf, lf f
197      if      test
198      db      '32'   ;32k example bios
199
200      endif
201      db      if      not test    ;memory size filled by relocator
202          db      '00'
203      db      'K CP/M vers '
204          db      ver/10+0', ', ' vers mod 10+0'
205          db      cr, lf, 0
206
207      boot:  ;print signon message and go to CCP
208          ;(note: mds boot initialized iobyte at 0003h)
209      lxi    sp, buff+80h
210      lxi    h, signon
211      call   Pmsg
212      xra   a
213      clraccr
214      setinitially
215      jmp   docpm
216
217      wboot: ;loader on track 0, sector 1, which will be skipped for warm
218          ;read CP/M from disk--assuming there is a 128 byte cold start
219          ;start
220
221      lxi    sp, buff
222      db      using dma--thus 80 thru ff available for stack

```

```

;                                ;      mvi    c, retry      imax retries
2222 16c6 0e0a          ;      push   b
2223 16c8 c5          ;boot0:    lxi    b, CPMB      ;set dma address to start of disk system
2224                                ;      call    setdma
2225                                ;      mvi    c, 0       ;boot from drive 0
2226 16c9 010000        ;      call    seldsk
2227 16cc cd8b17        ;      mvi    c, 0       ;start with track 0
2228 16cf 0e00          ;      call    settrk
2229 16d1 cd7d17        ;      mvi    c, 2       ;start reading sector 2
2230 16d4 0e00          ;      call    setsec
2231 16d6 cda717        ;      mvi    c, 0       ;read sectors, count nsects to zero
2232 16d9 0e02          ;      pop    b       ;10-error count
2233 16db cdac17        ;      mvi    b, nsects
2234                                ;      rdsec:   lxi    next sector
2235                                ;      push   b       ;read next sector
2236                                ;      mvi    b, nsects
2237 16df 0f62c          ;      push   b       ;save sector count
2238                                ;      read
2239 16e1 c5          ;      call    read
2240 16e2 cd8c17        ;      jnz    booterr
2241 16e5 c24917        ;      lhd    10d      ;retry if errors occur
2242 16e8 2a6b18        ;      lxi    d, 128
2243 16eb 118000        ;      dad    d       ;incremented dma address in hl
2244 16ee 19          ;      mov    b, h
2245 16ef 44          ;      mov    c, 1       ;ready for call to set dma
2246 16f0 4d          ;      call    setdma
2247 16f1 cd8b17        ;      lda    ios
2248 16f4 3a6b18        ;      cpi    26      ;sector number just read
2249 16f7 fela          ;      cpi    26      ;read last sector?
2250                                ;      jc    rd1
2251                                ;      must be sector 26, zero and go to next track

```

```

252    16fe 3a6a18      lda     int    ;get track to register a
253    16ff 3c          inr    a
254    1700 4f          mov    c, a   ;read for call
255    1701 cda717      call   setrk
256    1704 af          xra    a
257    1705 2c          rdi:   inr    a   ;clear sector number
258    1706 4f          mov    c, a   ;to next sector
259    1707 cdac17      call   setsec
260    170a c1          pop    b   ;recall sector count
261    170b 05          dcr    b   ;done?
262    170c c2e116      jnz    rdsec
263    ; done with the load, reset default buffer address
264    ; goCPM:           ;(enter here from cold start boot)
265    ; enable rst0 and rst7
266    170f f3          di
267    1710 3e12          mvi   a, 12h ;initialize command
268    1712 d3fd          out    revrt
269    1714 af          xra    a
270    1715 d3fc          out    intc ;cleared
271    1717 3e7e          mvi   a, inte ;rst0 and rst7 bits on
272    1719 d3fc          out    intc
273    171b af          xra    a
274    171c d3f3          out    icon ;interrupt control
275    ; set default buffer address to 80h
276    ; lxi   b, buff
277    171e 01B000      call   setdma
278    1721 cdbb17
279

```

```

280
281      ; reset monitor entry Points
282      l724 3ec3          mvi   a, jmp
283      l726 320000          sta   0
284      l729 210316          lxi   h, wboot
285      l72e 220100          shld  1      ;jump wboot at location 00
286      l72f 320500          sta   5
287      l732 210608          lxi   h, bdos
288      l735 220600          shld  6      ;jmp bdos at location 5
289      l738 323800          if    not test
290      l73b 2100f8          sta   7*8    ;jmp to mon80 (may have changed by ddtr)
291      l73e 223900          lxi   h, mon80
292      l73e 223900          shld  7*8+1
293      endif
294      leave iobyte set
295      ; previously selected disk was b, send parameter to cpm
296      l741 3a0400          lda   cdisk
297      l744 4f              mov   c, a      ;last logged disk number
298      l745 fb              ei
299      l746 c30000          jmp   cpmb
300
301      ; error condition occurred, print message and retry
302      booterr:           pop   b      ;recall counts
303      l749 c1              dcr   c
304      l74a 0d              jz    booterr
305      l74b ca5217          ; try again
306
307      l74e c5              push  b
308      l74f c3c916          jmp   wboot0

```

```

309
310      ; booter();
311      ; otherwise too many retries
312      lxi h, bootmsg
313      call PRMSG
314      jmp mon80  ; mds hardware monitor
315
316      ; bootmsg:
317      l75b 3f626f6f74  db  'Pboot', 0
318      ;
319      ;
320      const: console status to reg-a
321      ; (exactly the same as mds call)
322      1761 c312f8  jmp cst5
323
324      comin: iconsole character to reg-a
325      1764 cd03f8  call ci
326      1767 e67f  ani 7fh   ;remove Parity bit
327      1769 c9  ret
328
329      conout: iconsole character from c to console out
330      176a c309f8  jmp co
331
332      list: ilist device out
333      ; (exactly the same as mds call)
334      176d c30ff8  jmp lo
335
336      listst: ireturn list status
337      xra a
338      1770 af  ret    ; always not ready
339      1771 c9

```

```

340 ;          ; punch:   ipunch device out
341 ;          ; (exactly the same as mds call)
342          jmp    po
343 1772 c306f8 ;
344          ; reader:  ireader character in to reg-a
345          ; (exactly the same as mds call)
346          jmp    ri
347 1775 c306f8 ;
348          ; home:    imove to home position
349          ; treat as track 00 seek
350          jmp    r0
351 1778 0e00
352 177a c3a717 ;
353          ; seldisk: iselect disk given by register C
354          ; lxi    h, 0000h  ireturn 0000 if error
355 177d 210000
356 1780 79
357 1781 fe04
358 1783 d0 ;
359          ; ani    10b   i00 00 for drive 0, 1 and 10 10 for drive 2, 3
360 1784 e602
361 1786 3266f8
362 1789 79
363 178a e601
364 178c b7
365 178d ca9217
366 1790 3e30
367          ; sta    dbank  ito select drive bank
368          mov    a, c  i00, 01, 10, 11
369          ani    1b   imds has 0, 1 at 7B, 2, 3 at 88
370          ora    a   iresult 00?
371          jz    setdrive
372          mov    a, 00110000b  iselects drive 1 in bank

```

**setdrive:**

368 1792 47 mov b, a ; save the function  
369 1793 216818 lxi h, iof ; io function  
370 1796 7e mov a, m  
371 1797 e6cf ani 11001111b ; mask out disk number  
372 1799 b0 ora b ; mask in new disk number  
373 179a 77 mov m, a ; save it in iopb  
374 179b 69 mov l, c ;hl=disk number  
375 179c 2600 mvi h, 0  
376 179e 29 dad h ; \*2  
377 179f 29 dad h ; \*4  
378 17a0 29 dad h ; \*8  
379 17a1 29 dad h ; \*16  
380 17a2 113316 lxi d, dPbase  
381 17a5 19 dad d ;hl=disk header table address  
382 17a6 c9 ret  
383 ;  
384 ;  
385 ;

**setrk:** set track address given by C

386 17a7 216a18 lxi h, iot  
387 17aa 71 mov m, c  
388 17ab c9 ret

**setsec:** set sector number given by C

390 17ac 216b18 lxi h, ios  
391 17af 71 mov m, c  
392 17b0 c9 ret

```

394
395     17b1 0600          mvi   b, 0           ;translate sector bc using table at de
396     17b3 eb             xchg  b, d           ;idouble-Precision sector number in bc
397     17b4 09             dad   b               ;translate table address to hl
398     17b5 7e             mov   a, m           ;translate (sector) address
399     17b6 326b18         sta   105            ;translated sector number to a
400     17b9 6f             mov   a, l           ;return sector number in l
401     17ba c9             ret
402
403     ;setdma:    set dma address given by regs b, c
404     17bb 69             mov   l, c
405     17bc 60             mov   h, b
406     17bd 226c18         shld  iod
407     17c0 c9             ret
408
409     ;read:      read next disk record (assuming disk/trk/sec/dma set)
410     17c1 0e04           mvi   c, readf          ;set to read function
411     17c3 cde017         call   setfunc
412     17c6 cdf017         call   waitio          ;perform read function
413     17c9 c9             ret
414
415     ;write:     idisk write function
416
417     17ca 0e06           mvi   c, writf          ;set to write function
418     17cc cde017         call   setfunc
419     17cf cdf017         call   waitio          ;may have error set in reg-a
420     17d2 c9             ret
421

```

```

        ;
        ;
        ; utility subroutines
        ; Printmsg:    ; print message at h, l to 0
        422          mov    a, m
        423          ;                                ; i/o function address
        424          ;                                ; set it to accumulator for masking
        425          push   h
        426          17d3 7e      ;                                ; i/o function address
        427          17d4 b7      ;                                ; set it to accumulator for masking
        428          17d5 c8      ;                                ; i/o function address
        429          ;                                ; set it to accumulator for masking
        430          17d6 e5      ;                                ; i/o function address
        431          17d7 4f      ;                                ; set it to accumulator for masking
        432          17d8 cd6a17  ;                                ; i/o function address
        433          17d9 e1      ;                                ; set it to accumulator for masking
        434          17dc 23      ;                                ; i/o function address
        435          17dd c3d317 ;                                ; i/o function address
        436          jmp   printmsg
        ;
        ; Setfunc:     ; set function for next i/o (command in reg-c)
        437          ;                                ; i/o function address
        438          lxi   h, iof
        439          17e0 216818  ;                                ; i/o function address
        440          17e3 7e      ;                                ; set it to accumulator for masking
        441          17e4 e6f8      ;                                ; remove previous command
        442          17e5 b1      ;                                ; set to new command
        443          17e7 77      ;                                ; replaced in ioph
        444          ;                                ; the mds-800 controller requires disk bank bit in sector byte
        445          ;                                ; mask the bit from the current i/o function
        446          17e8 e620      ;                                ; i/o function address
        447          17ea 216618  ;                                ; mask the disk select bit
        448          17ed b6      ;                                ; address the sector select byte
        449          17ee 77      ;                                ; select proper disk bank
        450          17ef c9      ;                                ; set disk select bit on/off
        ret

```

```

451      ; waitio:
452      17f0 0e0a      mvi    c, retry      ;max retries before perm error
453      ; rewait:
454      ; start the i/o function and wait for completion
455      17f2 cd3f18      call   intype      ;in rtype
456      17f2 cd3f18      call   inbyte     ;clears the controller
457      17f5 cd4c18      ; 
458      17f8 3a6618      lda    dbank      ;set bank flags
459      17fb b7          ora    a           ;zero if drive 0, 1 and nz if 2, 3
460      17fc 3ee7          mvi   a, iopb      ;iopb and offh low address for iopb
461      17fe 0618          mvi   b, iopb      ;high address for iopb
462      1800 c20b18      shr   8           ;iopb shr 8
463      1803 d379          jnz   iodr1     ;idrive bank 1?
464      1803 d379          out   ilow       ;ilow address to controller
465      1805 78          mov   a, b       ;a, b
466      1806 d37a          out   ihigh      ;ihigh address
467      1808 c31018      jmp   waitio    ;to wait for complete
468      ; 
469      iodr1: idrive bank 1      ;idrive bank 1
470      180b d389      out   ilow+10h    ;ilow+10h for drive bank 10
471      180d 78          mov   a, b       ;a, b
472      180e d38a      out   ihigh+10h  ;ihigh+10h
473      ; 
474      1810 cd5918      waito: call  instat    ;wait for completion
475      1813 e604      ani   iordy    ;ready?
476      1815 ca1018      jz    waitio    ;waito

```

```

477      ; check i/o completion ok
478      1818 cd3f18      ; call intype      ; must be io complete (00) unlinked
479      00 unlinked i/o complete, 01 linked i/o complete (not used)
480      ; io disk status changed 11 (not used)
481      ; ready status change?
482      181b fe02      ; cpi 10b
483      181d ca3218      ; jz wready
484      ; must be 00 in the accumulator
485      1820 b7      ; ora a
486      1821 c23818      ; jnz werror      ; some other condition, retry
487      ; check i/o error bits
488      ; call inbyte
489      1824 cd4c18      ; ral
490      1827 17      ; jc wready      ; unit not ready
491      1828 da3218      ; rar
492      182b 1f      ; ani 1111110b      ; any other errors? (deleted data ok)
493      182c e6fe      ; jnz werror
494      182e c23818      ; read or write is ok, accumulator contains zero
495      ; ret
496      1831 c9      ; wready
497      ; not ready, treat as error for now
498      ; call inbyte      ; clear result byte
499      ; trycount
500      1832 cd4c18      ; jmp
501      1833 c33818
502      ;
```

```

503      ; error: !return hardware malfunction (crc, track, seek, etc.)
504      ;   the mds controller has returned a bit in each position
505      ;   of the accumulator, corresponding to the conditions:
506      ;     0   -deleted data (accepted as ok above)
507      ;     1   -crc error
508      ;     2   -seek error
509      ;     3   -address error (hardware malfunction)
510      ;     4   -data over/under flow (hardware malfunction)
511      ;     5   -write Protect (treated as not ready)
512      ;     6   -write error (hardware malfunction)
513      ;     j   -not ready
514      ;   (accumulator bits are numbered 7 6 5 4 3 2 1 0)
515
516
517      ; it may be useful to filter out the various conditions,
518      ; but we will get a permanent error message if it is not
519      ; recoverable, in any case, the not ready condition is
520      ; treated as a separated condition for later improvement
521
522      ; trycount:
523      ;   register c contains retry count, decrement 'til zero
524      ;   dc r    c
525      ;   jnz    rewait  ;for another try
526
527      ; cannot recover from error
528      ;   mvi    a, 1  ;error code
529      ;   ret

```

529  
530       ;       intype, inbyte, instat, read drive bank 00 or 10  
531     183f 3a6618     intype:     lda     dbank  
532     1842 b7              ora     a  
533     1843 c24918              jnz     intype1     ;skip to bank 10  
534     1846 db79              in     intype1  
535     1848 c9              ret     rtyPE  
536     1849 db89              intype1:     in     rtyPE+10h     ;B for 0, 1 BB for 2, 3  
537     184b c9              ret     ret  
538       ;  
539     184c 3a6618     inbyte:     lda     dbank  
540     184f b7              ora     a  
541     1850 c25618              jnz     inbyt1  
542     1853 db7b              in     inbyt1  
543     1855 c9              ret     rbyte  
544     1856 db8b              inbyt1:     in     rbyte+10h  
545     1858 c9              ret     ret  
546       ;  
547     1859 3a6618     instat:     lda     dbank  
548     185c b7              ora     a  
549     185d c26318              jnz     instal  
550     1860 db78              in     instal  
551     1862 c9              ret     dstat  
552     1863 db88              instat:     in     dstat+10h  
553     1865 c9              ret

```

554
555
556
557      1866 00      ; data areas (must be in ram)
558      dbank:    db     0      ; disk bank 00 if drive 0, 1
559                  ;          10 if drive 2, 3
560
561      1867 80      iopb:    i/o parameter block
562      1868 04      iof:    db     80h      ; normal i/o operation
563      1869 01      ion:    db     readf   ; i/o function, initial read
564      186a 02      iot:    db     1       ; number of sectors to read
565      186b 01      ios:    db     offset  ; track number
566      186c 8000      iod:    dw     1       ; sector number
567
568
569
570      186e+      ; define ram areas for bdos operation
571      186e+      bsegdat equ   $      ; directory access buffer
572      186e+      dirbuf: ds     128
573      18ee+      alv0:   ds     31
574      190d+      csv0:   ds     16
575      191d+      alv1:   ds     31
576      193e+      csv1:   ds     16
577      194e+      alv2:   ds     31
578      196e+      csv2:   ds     16
579      197e+      alv3:   ds     31
580      199a+      csv3:   ds     16
581      19aa=      enddat equ   $      ; -bsegdat
582      013c+      datsiz equ   $-bsegdat
583

```

als1	001f	141#
als2	001f	146#
als3	001f	151#
alv0	18ee	87
alv1	191d	91
alv2	194c	95
alv3	197b	99
base	0078	180#
bdd5	0896	29#
beddat	186e	571#
bias	0000	19#
boot	16bb3	63
booter0	1752	305
booterr	1749	241
bootmsg	175b	312
buff	0080	34#
cdisk	0004	33#
ci	f803	172#
co	f809	174#
conin	1764	66
conout	176a	67
const	1761	65
cplib	0000	28#
cplm1	1600	30#
cr	000d	192#
css1	0010	142#
css2	0010	147#
css3	0010	152#
csts	f812	177#
csv0	190d	87
csv1	193c	91

csv2	196b	95	578*
csv3	199a	99	580*
dat siz	013c	582#	
dbank	1866	361	459
di rbuf	186e	86	90
dPb0	1673	86	101#
dPb1	1673	90	140#
dPb2	1673	94	145#
dPb3	1673	98	150#
dPbase	1633	83#	380
dPe0	1633	84#	
dPe1	1643	88#	
dPe2	1653	92#	
dPe3	1663	96#	
dstat	0078	181#	550
endat	19aa	581#	
false	0000	15#	16
gocpm	170f	214	265#
home	1778	71	349#
icon	00fe	166#	275
ihidh	007a	186#	466
ilow	0079	185#	464
imbt1	1856	541	544#
inbyte	184c	457	490
instal	1863	549	552#
instsat	1859	474	547#
intc	00fc	165#	271
inte	007e	167#	272
intp1	1849	533	536#
intpe	183f	456	479
iod	186c	242	566#

iodd1	186b	463	469*
iof	1868	369	439
ion	1869	563#	562#
iopb	1867	461	462
lory	0004	191#	475
ios	186b	248	391
iot	186a	252	386
lf	000a	193#	196
list	176d	68	332*
listst	1770	78	336*
lo	f80f	176#	334
mon80	f800	170#	291
nsets	002c	31#	237
offset	0002	32#	100
Patch	1600	25#	27
po	f80c	175#	343
Prns g	1743	211	313
punch	1772	69	341#
rbyte	007b	183#	542
rd1	1705	250	257#
rdsec	16e1	238#	262
read	17e1	76	240
reader	1775	70	345#
readf	0004	188#	411
recal	0003	190#	411
retry	000a	35#	223
revit	00fd	164#	269
rewait	1712	454#	524
ri	f806	173#	347
mon80	ff0f	171#	314
rtypc	0079	182#	534

sectran	17b1	79	394*
seldisk	177d	72	229
setdma	17bb	75	227
setdrive	1792	365	367*
setfunc	17e0	412	419
setsec	17ac	74	233
settrk	17a7	73	231
signon	169c	195*	210
test	0000	16*	18
true	ffff	14*	15
trycount	1838	502	521*
vers	0016	6*	204
waito	1810	467	474*
waitio	17f0	413	420
wboot	16c3	64	217*
wboot0	16c9	225*	308
wboot0	1603	64*	294
werror	1838	487	495
wready	1832	483	492
write	17ca	77	417*
writf	0006	189*	418
xlt0	1682	84	112*
xlt1	1682	88	143*
xlt2	1682	92	148*
xlt3	1682	96	153*

L

L

L

L

L

L

L

L

L

L

L

L

L

L

L

L

# Appendix B

## A Skeletal CBIOS

```
1 ; skeletal cbios for first level of CP/M 2.0 alteration
2 ;
3     msize    equ    20          ;CP/M version memory size in Kilobytes
4 ;
5     ; "bias" is address offset from 3400h for memory systems
6     ; than 16K (referred to as "b" throughout the text)
7 ;
8     0000 = bias      (msize-20)*1024
9     3400 = ccp       3400h+bias           ;base of CCP
10    3c06 = bdos     ccp+806h            ;base of BDOS
11    4a00 = bios     ccp+1600h           ;base of BIOS
12    0004 = cdisk    0004h               ;current disk number 0=a,..., 15=d
13    0003 = iobYTE   0003h               ;intel i/o byte
14    ;
15    4a00      org    bios              ;origin of this program
16    002c = insects  ($-ccp)/128        ;warm start sector count
17    ;
18    ; JUMP vector for individual subroutines
19    4a00 c39c4a      jnp    boot             ;cold start
20    4a03 c3a64a      jnp    wboot            ;warm start
21    4a06 c3114b      jnp    const            ;console status
22    4a09 c3244b      jnp    conin            ;console character in
23    4a0c c3374b      jnp    conout           ;console character out
24    4a0f c3494b      jnp    list              ;list character out
25    4a12 c34d4b      jnp    punch             ;punch character out
```

```

26      4a15 c34f4b          ; reader character out
27      4a18 c3544b          ; move head to home position
28      4a1b c35a4b          ; select disk
29      4a1e c3744b          ; set track number
30      4a21 c3924b          ; set sector number
31      4a24 c3ad4b          ; set dma address
32      4a27 c3c34b          ; read disk
33      4a2a c3d64b          ; write
34      4a2d c34b4b          ; return list status
35      4a30 c3a74b          ; sector translate
36
37
38
39      4a33 734a0000          ; fixed data tables for four-drive standard
40      4a37 00000000          ; ibm-compatible 8" disks
41      4a3b f04c8d4a          ; disk Parameter header for disk 00
42      4a3f ec4d704d          ; dw trans, 0000h
43
44      4a43 734a0000          ; dw 0000h, 0000h
45      4a47 00000000          ; dw dirbf, dPblk
46      4a4b f04c8d4a          ; dw chk00, all00
47      4a4f fc4d8f4d          ; disk Parameter header for disk 01
48
49
50      4a53 734a0000          ; dw trans, 0000h
51      4a57 00000000          ; dw 0000h, 0000h
52      4a5b f04c8d4a          ; dw dirbf, dPblk
53      4a5f 0c4eaed4          ; dw chk02, all02

```

```

54      ; disk parameter header for disk 03
55      dw    trans, 0000h
56      dw    00000000
57      dw    dirbf, dfblk
58      dw    chk03, all03
59      ;
60      ; sector translate vector
61      dw    1, 7, 13, 19  ;sectors 1, 2, 3, 4
62      db    25, 5, 11, 17  ;sectors 5, 6, 7, 8
63      db    23, 3, 9, 15  ;sectors 9, 10, 11, 12
64      db    21, 2, 8, 14  ;sectors 13, 14, 15, 16
65      db    20, 26, 6, 12  ;sectors 17, 18, 19, 20
66      db    18, 24, 4, 10  ;sectors 21, 22, 23, 24
67      db    16, 22  ;sectors 25, 26
68      ;
69      ;idisk parameter block, common to all disks
70      dw    26  ;sectors per track
71      db    3  ;iblock shift factor
72      db    7  ;iblock mask
73      db    0  ;inull mask
74      dw    242  ;idisk size-1
75      dw    63  ;directory max
76      db    192  ;alloc 0
77      db    0  ;alloc 1
78      dw    16  ;check size
79      dw    2  ;track offset
80      ;

```

; end of fixed tables

```

82
83 ; individual subroutines to perform each function
84 boot:           ; isimplest case is to just perform parameter initialization
85     4a9c af      xra    a          ; zero in the accum
86     4a9d 320300   sta    iobyte   ; clear the iobyte
87     4aa0 320400   sta    cdisk   ; select disk zero
88     4aa3 c3ef4a   jmp    gocPM  ; initialize and go to CP/M
89
90 ;wboot:           ; isimplest case is to read the disk until all sectors loaded
91     4aa6 318000   lxi    sp, 80h  ; use space below buffer for stack
92     4aa9 0e00   mvi   c, 0       ; select disk 0
93     4aab cd5a4b   call   seldsk ; call seldsk
94     4aae cd544b   call   home   ; go to track 00
95
96     4ab1 062c   mui   b, nsects ; b counts # of sectors to load
97     4ab3 0e00   mvi   c, 0       ; c has the current track number
98     4ab5 1602   mvi   d, 2       ; d has the next sector to read
99
100    ; note that we begin by reading track 0, sector 2 since sector 1
101    ; contains the cold start loader, which is skipped in a warm start
102    4ab7 210034   lxi    h, ccp  ; CCP base of CP/M (initial load point)
103    4aba c5      push   b       ; save sector count, current track
104    4abb d5      push   d       ; save next sector to read
105    4abc e5      push   h       ; save DMA address
106    4abd 4a      mov    c, d   ; get sector address to register C
107    4abe cd924b   call   setsec ; set sector address from register C
108    4ac1 c1      pop    b       ; recall DMA address to B, C
109    4ac2 c5      push   b       ; replace on stack for later recall
110    4ac3 cdaddb   call   setdma ; set DMA address from B, C

```

```

111          ; drive set to 0, track set, sector set, dma address set
112          call read
113          cdc34b
114          fe00
115          c2a64a
116          jnz kboot
117          ; iretry the entire boot if an error occurs
118          dace e1
119          4acf 118000
120          4ad2 19
121          4ad3 d1
122          4ad4 c1
123          4ad5 05
124          4ad6 caef4a
125          ; no error, move to next sector
126          pop h
127          1xi d, 128
128          dad d
129          pop d
130          pop b
131          dcr b
132          jz $OCPM
133          ; more sectors remain to load, check for track change
134          ; isector=27?, if so, change tracks
135          ; icarry generated if sector<27
136          ; end of current track, go to next track
137          ; mvf d, 1
138          ; ibegin with first sector of next track
139          ; itrack=track+1

```

```

135
136      ; save register state, and change tracks
137      4ae3 c5      push b
138      4ae4 d5      push d
139      4ae5 e5      push h
140      4ae6 cd7d4b   call settrk
141      4ae9 e1      pop h
142      4aea d1      pop d
143      4aeb c1      pop b
144      4aec c3bada   jmp load    ;for another sector
145
146      ; end of load operation, set parameters and go to CP/m
147      jocpm:
148      4aef 3ec3     mvi a, 0c3h   ;c3 is a jmp instruction
149      4af1 320000   sta 0       ;for jmp to wboot
150      4af4 21034a   lxi h, wboote ;wboot entry point
151      4aff 220100   shld 1     ;set address field for jmp at 0
152
153      4afa 320500   ;for jmp to bdos
154      4af9 21063c   sta 5       ;bdos entry point
155      4b00 220600   lxi h, bdos ;address field of jump at 5 to bdos
156
157      4b03 018000   lxi b, 80h  ;default dma address is 80h
158      4b06 cdaddb   call setdma ;
159
160      4b09 fb      ei
161      4b0a 3a0400   lda cdisk
162      4b0d 4f      mov c, a
163      4b0e c30034   jmp CCP

```

enable the interrupt system  
 ;get current disk number  
 ;send to the CCP  
 ;go to CP/m for further processing

```

164
165 ; simple i/o handlers (must be filled in by user)
166 ; in each case, the entry point is provided, with space reserved
167 ; to insert your own code
168 ;
169
170 const:  iconsole status, return 0fh if character ready, 00h if not
171         ds 10h      ;space for status subroutine
172         mvi a, 00h
173         ret
174
175 conin:  iconsole character into register a
176         ds 10h      ;space for input routine
177         ani 7fh    ;strip parity bit
178         ret
179
180 conout: iconsole character output from register c
181         mov a, c    ;get to accumulator
182         ds 10h      ;space for output routine
183         ret
184
185 list:   ;list character from register c
186         mov a, c    ;character to register a
187         ret
188
189 listst: ;return list status (0 if not ready, 1 if ready)
190         xra a      ;a is always OK to return
191         ret

```

```

192 ;          ;
193 punch:    ;Punch character from register C
194     mov    a, c      ;character to register a
195     ret             ;null subroutine
196 ;
197 ;
198 reader:   ;reader character into register a from reader device
199     mvi   a, 1ah      ;enter end of file for now (replace later)
200     ani   7fh        ;remember to strip Parity bit
201     ret             ;
202 ;
203 ;
204 i/o drivers for the disk follow
205 ; for now, we will simply store the parameters away for use
206 ; in the read and write subroutines
207 ;
208 home:    ;move to the track 00 position of current drive
209 ; translate this call into a settrk call with Parameter 00
210     mvi   c, 0      ;select track 0
211     call  settRk      ;we will move to 00 on first read/write
212     ret             ;

```

```

213
214     ; selDisk:  iselect disk given by register C
215     4b51 210000  lxi    h, 0000h  ;error return code
216     4b5d 79      mov    a, c
217     4b5e 32ef4c  sta    diskno
218     4b61 fe04  cpi    4          ;must be between 0 and 3
219     4b63 d0      mc    ;no carry if 4, 5, ...
220
221     4b64          ; disk number is in the proper range
222
223     4b6e 3aef4c  ds    10        ;space for disk select
224     4b71 6f      lda    diskno
225     4b72 2600  mov    l, a        ;l=disk number 0, 1, 2, 3
226     4b74 29      movi   h, 0        ;high order zero
227     4b75 29      dad    h
228     4b76 29      dad    h        ;*2
229     4b77 29      dad    h        ;*4
230     4b78 11334a  lxi    d, dPbase
231     4b7b 19      dad    0        ;hl=dPbase (diskNo*16)
232     4b7c c9      ret
233
234     4b7d 79      ; setTrk:  iset track given by register C
235     4b7e 32e94c  mov    a, c
236     4b81          sta    track
237     4b81          ds    10h        ;space for track select
238     4b91 c9      ret

```

```

239
240      ; setsec:   ;set sector given by register c
241      4b92 79    mov    a, c
242      4b93 32eb4c  sta    sector
243      4b96          ds    10h   ;space for sector select
244      4ba5 c9      ret
245
246      ; section:
247      ; translate the sector given by bc using the
248      ; translate table given by de
249      4ba7 eb      xchg   ihl=,trans
250      4bab 09      dad    b
251      4ba9 6e      mov    l, m
252      4baa 2600    mvi   h, 0
253      4bac c9      ret
254
255      ; setdma:   ;set dma address given by registers b and c
256      4bad 69      mov    1, c
257      4bae 60      mov    h, b
258      4baf 22ed4c  shld   dmaad
259      4bbf 22        ds    10h   ;space for setting the dma address
260      4bc2 c9      ret
261
262      ; read:    ;perform read operation (usually this is similar to write
263      ; so we will allow space to set up read command, then use
264      ; common code in write)
265      4bc3          ds    10h   ;set up read command
266      4bd3 c3e64b  jmp   waitio  ;to perform the actual i/o
267
268      ; write:   ;perform a write operation
269      4bd6          ds    10h   ;set up write command

```

```

270
271    ; enter here from read and write to perform the actual i/o
272    ; operation. return a 00h in register a if the operation completes
273    ; properly, and 01h if an error occurs during the read or write
274
275    ; in this case, we have saved the disk number in 'diskno' (0, 1)
276    ; the track number in 'track' (0-76)
277    ; the sector number in 'sector' (1-26)
278    ; the dma address in 'dmaad' (0-65535)
279    ; ispace reserved for i/o drivers
280    ; error condition
281    ; replaced when filled-in
282
283    ; the remainder of the cbios is reserved uninitialized
284    ; data area, and does not need to be a part of the
285    ; system memory image (the space must be available,
286    ; however, between "begdat" and "enddat").
287
288    track: ds   2   ; two bytes for expansion
289    sector: ds  2   ; two bytes for expansion
290    dmaad: ds  2   ; indirect memory address
291    diskno: ds  1   ; disk number 0-15

```

```
292
293      ; scratch ram area for bdos use
294      4cf0=    begdat    equ     $          ;beginning of data area
295      4cf0      dirfb:   ds      128       ;scratch directory area
296      4d70      al100:   ds      31        ;allocation vector 0
297      4dbf      al101:   ds      31        ;allocation vector 1
298      4dae      al102:   ds      31        ;allocation vector 2
299      4dc0      al103:   ds      31        ;allocation vector 3
300      4dec      chk00:   ds      16        ;check vector 0
301      4dfc      chk01:   ds      16        ;check vector 1
302      4e0c      chk02:   ds      16        ;check vector 2
303      4e1c      chk03:   ds      16        ;check vector 3
304      ;           enddat   equ     $          ;end of data area
305      4e2c      datsiz  equ     $-begdat; ;size of data area
306      013c=    end
307      4e2c
```

a1100	4d70	43	296#
a1101	4dBf	48	297#
a1102	4dae	53	298#
a1103	4dcf	58	299#
bios	3c06	10#	154
boot	4cf0	294#	306
ccP	3400	9#	10
cdisk	0004	12#	87
chk00	4dec	43	300#
chk01	4dfc	48	301#
chk02	4e0c	53	302#
chk03	4e1c	58	303#
conin	4b24	22	173#
conout	4b37	23	180#
const	4b11	21	170#
datasiz	013c	306#	
dirbf	4cf0	42	47
diskio	4cef	217	223
dmaad	4ced	258	290#
database	4a33	40#	
dpblk	4a8d	42	47
enddat	4e2c	305#	
goopm	4aef	88	124
home	4b54	27	94
iobyte	0003	13#	86
list	4b49	24	185#
listst	4b4b	34	189#
load1	4aba	102#	130

msize	0014	3#	8
nsects	002c	16#	96
Punch	4b4d	25	193#
read	4bc3	32	113
reader	4b4f	26	198#
sector	4ceb	242	289#
sectran	4ba7	35	246#
seldisk	4b5a	28	93
setdma	4bad	31	110
setsac	4b92	30	107
settrk	4b7d	29	140
track	4ce9	236	288#
trans	4a73	40	45
waitio	4be6	266	271#
wboot	4aa6	20	90#
wboote	4a03	20*	150
write	4bd6	33	268#

*End of Appendix B*

## Appendix C

### A Skeletal GETSYS/PUTSYS Program

combined getsys and putsys programs from

Sec 6.4

Start the programs at the base of the TPA

```
org 0100h

0014 =                         ; size of CP/m in Kbytes
                                ; "bias" is the amount to add to addresses for > 20K
                                ; (referred to as "b" throughout the text)

                                ; bias
                                ; equ (msize-20)*1024
                                ; CCP
                                ; equ 3400h+bias
                                ; equ CCP+8000h
                                ; equ CCP+16000h
                                ; getsys programs tracks 0 and 1 to memory at
                                ; 38800h + bias

0000 =                         ; bias
3400 =                         ; CCP
3000 =                         ; BIOS
4000 =                         ; ;
```

```

; register usage
;   a      (scratch register)
;   b      track count (0...76)
;   c      sector count (1...26)
;   d,e   (scratch Register Pair)
;   h,l   load address
;   sp    set to track address

; start of getsys
gstart:    lxi sp,ccp-00B0h ;convenient Place
            lxi h,ccp-00B0h ;set initial load
            mvi b,0           ;start with track
            rd$trk:          ;read next track
            mvi c,1           ;each track start

rd$sec:    call read$sec ;get the next sector
            lxi d,128         ;offset by one sector
            dad d             ;(hl=hl+128)
            inr c             ;next sector
            mov a,c           ;fetch sector number
            cpi 27            ;and see if last
            jc rd$sec         ;<, do one more

; arrive here at end of track, move to next track

            inr b             ;track = track+1
            mov a,b           ;check for last
            cpi 2             ;track = 2 ?
            jc rd$trk         ;<, do another

```

0100 31B033  
0103 21B033  
0106 0600  
0108 0e01  
010a cd0003  
010d 11B000  
0110 19  
0111 0c  
0112 79  
0113 fe02  
0115 da0a01

arrive here at end of load, halt for lack of anything  
better .

```
011f  fb          ei
0120 76          hlt
                  ; Putsys Program, Places memory image
                  ; starting at
                  ; 3880h + bias back to tracks 0 and 1
                  ; start this program at the next page boundary
0200              org ($+0100h) and 0ff00h

Put$sys:
0200  31B033      lxi sp,ccp-0080h  ;inconvenient place
0203  21B033      lxi h,ccp-0080h  ;start of dump
0206  0600          mvi b,0           ;start with track

wr$trK:
0208  0e01          mvi b,1           ;start with sector

wr$sec:
020a  cd0004      call write$sec   ;write one sector
020d  11B000      lxi d,128     ;length of each
0210  19          dad d        ;i<1>=<1> + 128
0211  0c          inn c        ;i <c>=<c> + 1
0212  79          mov a,c      ;see if
0213  fe1b          cpi 27      ;past end of track
0215  data02      jc wr$sec    ;no, do another

arrive here at end of track, move to next track
```

```

0218 04    inn b      i$track = track+1
0219 78    mov a,b    i$ee if
021a fe02    cpi 2      i$last track
021c da802   jc wr$trk  i$no, do another

; done with putsys, halt for lack of anything
; better

021f fb    ei
0220 76    hit

; user supplied subroutines for sector read and write

; move to next page boundary
; org ($+0100h) and 0F00h

read$sec:
; read the next sector
; i$track in <b>,
; i$sector in <c>
; idmaaddr in <hl>

0300 05    push b
0301 e5    push h

; user defined read operation goes here
; ds 64

0302

```

0342 e1  
0343 c1  
0344 c9

0400

POP h  
POP b  
ret

org (\$+0100h) and off00h ;another page  
;boundary

write\$sec:

;same parameters as read\$sec

0400 c5  
0401 e5

PUSH b  
PUSH h

;user defined write operation goes here

0402

ds 64

0442 e1  
0443 c1  
0444 c9

POP h  
POP b  
ret

;end of setsys/putsys program

0445

end

End of Appendix C

Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ Γ

# Appendix D

## The MDS-800 Cold Start Loader for CP/M 2.0

```
1      title mds cold start loader at 3000h'
2
3      ; mds-800 cold start loader for CP/m 2.0
4
5      ; version 2.0 august, 1979
6
7      0000 =          false    equ    0
8      ffff =          true     equ    not false
9      0000 =          testing  equ    false   if true, then go to mon80 on errors
10
11     if      testing
12     bias   equ    03400h
13     endif
14     if      not testing
15     bias   equ    0000h
16     endif
17     0000 =          cpmb   equ    bias
18     0806 =          bdos   equ    806h+bias
19     1880 =          bdose  equ    1880h+bias
20     1600 =          boot   equ    1600h+bias
21     1603 =          rboot  equ    boot+3
                                bias   equ    base of dos load
                                entry to dos for calls
                                end of dos load
                                icold start entry Point
                                iwarm start entry Point
```

```

22      30000          ; loaded down from hardware boot at 3000H
23
24      ;                                org    03000h
25      bddos1    equ     bddose-cPMB
26      nttrks   equ     2           ; number of tracks to read
27      bdos1    equ     bdos1/128 ; number of sectors in dos
28      bdos0    equ     25          ; number of bdos sectors on track 0
29      bdos1    equ     bdos5-bdoso
30
31      mon80    equ     0f800h   ; intel monitor base
32      mon80    equ     0ff0fh   ; restart location for mon80
33      base     equ     078h     ; 'base' used by controller
34      rtype    equ     base+1   ; result type
35      rbyte   equ     base+3   ; result byte
36      reset    equ     base+7   ; reset controller
37
38      dstat    equ     base     ; idisk status Port
39      ilow    equ     base+1   ; ilow iopb address
40      ihigh   equ     base+2   ; ihigh iopb address
41      bsw     equ     0ffh     ; iboot switch
42      00f?    equ     recal   ; recalibrate selected drive
43      0003    equ     3h       ; readf read function
44      0004    equ     4h       ; stack end of boot for stack
45
46      0100    equ     100h
47      30000 3100001          ; lxi sp,stack; ; in case of call to mon80
48
49      3003  db79            ; clear disk status
50      3005  db7b            ; in rtype
51

```

```

52      coldstart:          in      bsw
53      3007 dbff          ani     02h      ;switch on?
54      3009 e602          jnz     coldstart
55      300b c20730         ; clear the controller
56      300e d37f          out    reset      ;logic cleared
57      300e d37f          ; 
58      300e d37f          ; 
59      3010 0602          movi   b,ntrks      ;number of tracks to read
60      3012 214230         lxi   h,iopbo
61      ; 
62      ; 
63      start:             ; 
64      ; 
65      ; read first/next track into CPMB
66      3015 7d             mov    a,l
67      3016 d379           out    ilow
68      3018 7c             mov    a,h
69      3019 d37a           out    ihigh
70      301b db78           waito: in      dstat
71      301d e604           ani    4
72      301f ca1b30         jz    waito
73      ; 
74      ; check disk status
75      3022 db79           in      rt/rp
76      3024 e603           ani    11b
77      3026 fe02           cpi    2
78      ; 
79      if      testing
80      cnc   rmon80      ;go to monitor if 11 or 10
81      endif

```

```

82      3028 d20030          ; not testing
83      jnc    rstart         ; retry the load
84      endif
85      ;                                ; if
86      302b db7b          ;   in      rbyte      ; i/o complete, check status
87      ;   if not ready, then go to mon80
88      302d 17          ;   ral
89      302e dc0fff        ;   cc      rmonB0      ;not ready bit set
90      3031 1f          ;   rar
91      3032 e61e          ;   ani     11110b      ;overrun/addr err/seek/crc/xxx
92      ;                                ; if
93      testing           ;   testing
94      rmonB0            ;   cnz      rgo to monitor
95      endif
96      ;                                ; if
97      3034 c20030        ;   if      not testing
98      jnz    rstart         ;   rstart ; retry the load
99      ;                                ; endif
100     3037 110700        ;   ;      lxi      d,iopb1      ;length of iopb
101     ;      dad      d      addressing next iopb
102     303a 19          ;   dcr      b      ;count down tracks
103     303b 05          ;   jnz
104     303c c21530        ;   start
105     ;                                ; if
106     303f c30016        ;   ;      jmp    to boot to print initial message, and set up jmps
107     ;      jmp    boot
108     ;                                ; if
109     ;      jmp    Parameter blocks
110     3042 80          ;   iopb:  db      80h      ;iowc, no update
111     3043 04          ;   db      db      ;readf
112     ;                                ; read function

```

```

113      3044 19          db      bd050    i#sectors to read on track 0
114      3045 00          db      0        itrack 0
115      3046 02          db      2        start with sector 2 on track 0
116      3047 0000          dw      CPMB    start at base of bdos
117      0007 =           ioPb1   equ     $-ioPb0
118      ;                  ;
119      3049 80          ioPb1:  db      80h    readf
120      304a 04          db      bd051    i#sectors to read on track 1
121      304b 18          db      1        itrack 1
122      304c 01          db      1        isector 1
123      304d 01          db      cmpb+bd050*128ibase of second read
124      304e 800c          dw
125      ;                  ;
126      3050          end

base      0078    33#    34      35      36      38      39      40
bdos     0806    18#    18#    29#    29      113     124
bd050    0019    28#    28#    29#    121     19#
bd051    0018    29#    29#    121     25      25
bdose    1880    1880   1880   1880   1880   19#    21      108
bd051    0031    0031   0031   0031   0031   25#    27
bd055    0000    0000   0000   0000   0000   27#    29
bias     1600    1600   1600   1600   1600   12#    15#    17      18      19      20
boot     0fff     0fff   0fff   0fff   0fff   20#    21      108
bsw      coldstart 3007    3007   52#    52#    55      116     124
CPMB    0000    0000   0000   0000   0000   17#    17#    25
dstat    0078    0078   0078   0078   0078   38#    70
false    0000    0000   0000   0000   0000   7#     8       9
high    007a    007a   007a   007a   007a   40#    40#    69

```

low	0079	39*	67
lpbo	3042	61	111*
lpb1	3049	119*	
lpb1	0007	101	117*
mon80	f800	31*	
ntks	0002	26*	60
rboot	1603	21*	
rbyte	007b	35*	50
readf	0004	43*	120
recall	0003	42*	
reset	007f	36*	57
mon80	ff0f	32*	80
restart	3000	46*	83
rtype	0079	34*	49
stack	0100	44*	47
start	3015	63*	104
testing	0000	9*	11
true	ffff	8*	
waito	301b	70*	72

*End of Appendix D*

# Appendix E

## A Skeletal Cold Start Loader

This is a sample cold start loader, which, when modified resides on track 00, sector 01 (the first sector on the diskette). We assume that the controller has loaded this sector into memory upon system start-up (this program can be keyed-in, or can exist in read-only memory beyond the address space of the CP/M version you are running), the cold start loader brings the CP/M system into memory at "loadp" (3400h + bias), in a 20K memory system, the value of "bias" is 000h, with large values for increased memory sizes (see section 2). After loading the CP/M system, the cold start loader branches to the "boot" entry point of the BIOS, which begins at :"bios" + "bias". The cold start loader is not used until the system is powered up again, as long as the BIOS is not overwritten, the origin is assumed at 0000h, and must be changed if the controller brings the cold start loader into another area, or if a read-only memory area is used.

```

0000          org 0          ibase of ram in
                           CCP/m

0014 =      msizE     equ 20          imin mem size in
                           bytes
                           offset from 20K
                           system

0000 =      bias       equ (msize-20)*1024
                           CCP
                           bios     equ 34000h+bias
                           bios1    equ 0300h
                           boot    equ bios
                           size     equ bios+bios1-ccP
                           sects   equ size/128
                           # of sectors to load

                           ; begin the load operation

0032 =      cold:
                           lxi b,2          ib=0, c=sector 2
                           movi d,sectS      id=# sectors to
                           load
                           lxi h,ccP          ibase transfer
                           address
                           lsect:            iload the next sector

                           ; insert inline code at this point to
                           ; read one 128 byte sector from the
                           ; track given in register b, sector
                           ; given in register c,
                           ; into the address given by <hl>
                           ; branch to location "cold" if a read error occurs

```

```
0008 c36b00
000b

006b 15
006c ca004a

006f 318000
0072 39
0073 0c
0074 79

; user supplied read operation goes
; here...
;
; jmp Past$Patch           ;remove this
;                                ;when Patched
; ds    60h

Past$Patch:
; go to next sector if load is incomplete
        dcr d      ;sector=sectors-1
        jz boot   ;boot head for the bios
;
; more sectors to load
;
; we aren't using a stack, so use <sp> as scratch
; register
; to hold the load address increment
        lxi sp,128  ;128 bytes per
;sector
        dad sp      ;<hl> = <hl> +
128
        inr c      ;sector=sector + 1
        mov a,c
```

```
0075 f0lb          ;last sector of
                    ;it track?
cpi 27
0077 da0800        ;no, go read
                    ;another

                    ;end of track, increment to next track

007a 0e01          ;sector = 1
007c 04          ;it track = track + 1
007d c30900        ;for another group
0080
                    ;of boot loader

                    ;end of track, increment to next track

007a 0e01          ;sector = 1
007c 04          ;it track = track + 1
007d 1sect         ;for another group
0080
                    ;of boot loader
```

*End of Appendix E*

# Appendix F

## CP/M Disk Definition Library

This listing has been printed broadside on the page, so that you can easily read the entire listing.

```
1:: CP/M 2.0 disk re-definition library
2:: 21:: Copyright (c) 1979
3:: 4:: Digital Research
4:: Box 579
5:: 6:: Pacific Grove, CA
6:: 7:: 93550
7:: 8:: CP/M logical disk drives are defined using the
8:: macros given below, where the sequence of calls
9:: is:
9:: 10:: disks n
10:: 11:: diskdef Parameter-list-0
11:: 12:: diskdef Parameter-list-1
12:: 13:: ...
13:: 14:: ...
14:: 15:: ...
15:: 16:: ...
16:: 17:: ...
17:: 18:: ...
18:: 19:: ...
19:: 20:: ...
20:: 21:: ...
21:: 22:: ...
```

where n is the number of logical disk drives attached to the CP/M system, and Parameter-list-i defines the characteristics of the ith drive (i=0,1,...,n-1)

```
23:#  
24:#  
25:#  
26:#  
27:#  
28:#  
29:#  
30:#  
31:#  
32:#  
33:#  
34:#  
35:#  
36:#  
37:#  
38:#  
39:#  
40:#  
41:#  
42:#  
43:#  
44:#  
45:#  
46:#  
47:#  
48:#  
49:#  
50:#  
51:#
```

each parameter-list-i takes the form

```
dn,fsc,lsc,[skf],blk,dk,cks,ofs,[0]
```

where

dn	is the disk number 0,1,...,n-1
fsc	is the first sector number (usually 0 or 1)
lsc	is the last sector number on a track
skf	is optional "skew factor" for sector translate
blk	is the data block size (1024/2048,...,16KB)
dk	is the disk size in bbs increments (word)
dir	is the number of directory elements (word)
cks	is the number of dir elements to checksum
ofs	is the number of tracks to skip (word)
[0]	is an optional 0 which forces 16K/directory end

for convenience, the form

```
dn,dm
```

defines disk dn as having the same characteristics as

a previously defined disk dm.

a standard four drive CP/M system is defined by

disks	4
diskdef	0,1,26,6,1024,243,64,64,2
dsk	set 0
rept	3
dsk	set dsk+1
diskdef	%dsk,0
endm	
endef	

```

52: ;  

53: ; the value of "begdat" at the end of assembly defines the  

54: ; beginning of the uninitialized ram area above the bios,  

55: ; while the value of "enddat" defines the next location  

56: ; following the end of the data area, the size of this  

57: ; area is given by the value of "datasz" at the end of the  

58: ; assembly, note that the allocation vector will be quite  

59: ; large if a large disk size is defined with a small block  

60: ; size.  

61:;  

62: dsKhdr  

63: ij  

64: dpe&dn:  

65:  

66:  

67:  

68:  

69:;  

70: disks  

71: ij;  

72: ndisks  

73: ddbase  

74: ij;  

75: disknxt  

76:  

77:  

78: disknxt  

79:  

80:;

macro      dn
define    a single disk header list
          xlt&dn,0000h      ;translate table
          0000h,0000h        ;scratch area
          dirbuf,dpb&dn     ;dir buff,parm block
          csv&dn,alv&dn      ;check, alloc vectors
endm

macro      nd
define    nd disks
set      nd
equ      $           ;generate the nd elements
generate
set      0
rept      nd
dsKhdr  %disknxt
set      disknxt+1
endm

```

```

81;;
82:dbbhdr          macro   dn
83:dpb&dn          equ     $
endm               idisk param block

85;;
86:ddb             macro   data,comment
87:ii              define  a db statement
88:                db     data
89:                endm               comment

90;;
91:ddw             macro   data,comment
92:ii              define  a dw statement
93:                dw     data
94:                endm               comment

95;;
96:gcd             macro   m,n
97:ii              greatest common divisor of m,n
98:ii              produces value gcdn as result
99:ii              (used in sector translate table generation)
100:gcdm            set    m
101:gcdn            set    n
102:gcdr            set    0
103:                rept   65535
104:gcdx            set    gcdm/gcdn
105:gcdr            set    gcdm-gcdx*gcdn
106:                if    gcdr = 0
107:                exitm
108:                endif
109:gcdm            set    gcdn
110:gcdn            set    gcdr
111:                endm

```

```

112:
113;;
114:diskdef
115;;
116:
117;;
118:dpb&dn
119:als&dn
120:css&dn
121:xlt&dn
122:
123:secmax
124:sectors
125:als&dn
126:
127:als&dn
128:
129:css&dn
130;;
131:blkval
132:blkshf
133:blkmsk
134:
135:
136:
137:
138;;
139:blkshf
140:blkmsk
141:blkval
142:

endm

macro dn,fsc,lsc,skf,bls,dk5,dr5,cks,ofs,k16
    generate the set statements for later tables
        if nul lsc
            same as Previous fsc
        current disk dn
            dpb&fsc
        equ
            als&fsc
        equ
            css&fsc
        equ
            xlt&fsc
        else
            lsc-(fsc)
            set secmax
            ;sectors 0..secmax
            set sectors+1
            ;number of sectors
            set (dk5)/8
            ;size of allocation vector
            if ((dk5)mod8) ne 0
                set als&dn+1
            endif
            set (cks)/4
            ;number of checksum elements
            generate the block shift value
            set bls/128
            ;number of sectors/block
            set 0
            ;counts right 0's in blkval
            set 0
            ;fills with 1's from right
            rept 16
            ;once for each bit position
            if blkval=1
                exitm
            endif
            otherwise, high order 1 not found yet
            set blkshf+1
            set (blkmsk shr 1) or 1
            set blkval/2
endm

```

```

143;;
144blkval
145extmsk
146;;
147:
148:
149:
150;;
151:extmsk
152:blkval
153:
154;;
155:
156extmsk
157:
158;;
159:
160extmsk
161:
162;;
163:dirrm
164:dirbk
165:dirblk
166:
167:
168:
169:

generate the extent mask byte
set    b1s/1024      ;number of Kilobytes/block
set    0              ;fill from right with 1's
rep    16
if    blkval=1
exitm
endiff
otherwise more to shift
set    (extmsk shr 1) or 1
set    blkval/2
endm
may be double byte allocation
if    (dk5)>256
set    (extmsk shr 1)
endiff
may be optional [0] in last position
if    not nul K16
set    K16
endiff
now generate directory reservation bit vector
set    dir      ;#remaining to process
set    b1s/32     ;number of entries per block
set    0          ;fill with 1's on each loop
rep    16
if    dirrem=0
exitm
endiff

```

```

170::: not complete, iterate once again
171::: shift right and add 1 high order bit
172:dirblk set (dirblk shr 1) or 8000h
173: if dirrem>dirblk
174: set dirrem-dirblk
175: else
176:direm set 0
177:endif
178:endifm
179: dPbndr dn ;generate equ $
180: ddw %sectors,<isec per track>
181: ddb %blkshf,<iBLOCK shift>
182: ddb %blkmsk,<iBLOCK mask>
183: ddb %extmsk,<extn mask>
184: ddw %(dks)-1,<iDisk size-1>
185: ddw %(dir)-1,<directory max>
186: ddb %dirblk shr 8,<ialloc0>
187: ddb %dirblk and Ofh,<ialloc1>
188: ddw %cks)/4,<iCheck size>
189: ddw %ofs,<offset>
190::: generate the translate table, if requested
191: if nul $kf
192:xlt&dn equ 0 ;no xlate table
193: else
194: if $kf = 0 ;no xlate table
195:xlt&dn equ 0
196: else
197::: generate the translate table
198:nxtsec set 0 ;next sector to fill
199:nxtbas set 0 ;moves by one on overflow
200:gcd

```

```

201:;;
202:neltst
203:;;
204:;;
205:nelts
206:xlt&dn
207:;
208:;
209:;
210:;
211:;
212:;
213:nxtsec
214:;
215:nxtsec
216:;
217:nelts
218:;
219:nxtbas
220:nxtsec
221:nelts
222:;
223:;
224:;
225:;
226:;
227:;;
228:ndefs
229:lab:
230:;

gcdn = gcd(sectors,skew)
set sectors/gcdn
neltst is number of elements to generate
before we overlap previous elements
set neltst
      if
      sectors < 256
      ddb %nxtsec+(fsc)
      else
      ddw %nxtsec+(fsc)
endiff
set nxtsec+(skf)
if nxtsec>=sectors
set nxtsec-sectors
endiff
set neltst1
if neltst = 0
set nxtbas+1
set nxtbas
set neltst
endiff
endifm
endiff
endiff
endifm
macro lab space
ds space
endifm

```

```

231:;  

232:lds          macro    lb,ds,ual  

233:            defds   lb&ds,%ual&dn  

234:            endm  

235:;  

236:edef          macro   generate the necessary ram data areas  

237:;;  

238:besdat        equ     $           128          ;directory access buffer  

239:dirbuf:       ds      128  

240:dsKnxT        set     0           ;number of disks  

241:              rept   ndisks  ;one for each disk  

242:              lds    alv,%dsKnxT,alls  

243:              lds    osv,%dsKnxT,ccs  

244:dsKnxT        set     dsKnxT+1  

245:              endm  

246:enddat        equ     $           $-begdat  

247:datsiz       equ     db 0           ;at this point forces hex record  

248:;;  

249:            endm

```

*End of Appendix F*

ר

ר

ר

ר

ר

ר

ר

ר

ר

# Appendix G

## Blocking and Deblocking Algorithms

This listing has been printed broadside on the page, so that you can easily read the entire listing.

```
1      sector deblocking algorithms for CP/M 2.0
2
3
4
5
6
7      utility macro to compute sector mask
8      Macro          hblk
9      // compute log2(hblk), return @x as result
10     // (2 ** @x = hblk on return)
11     set            hblk
12     set            0
13     count right shifts of @y until = 1
14     rep#           8
15     if             @y = 1
16     exitm
17     endif
18     // @y is not 1, shift right one position
19     set            @y shr 1
20     set            @x + 1
21
22     endm
endm
```

23  
24  
25  
26  
27

CP/m to host disk constants

```
28 ;  
29 0800 = blksiz    equ      2048    ;cp/m allocation size  
30 0200 = hstsiz    equ      512     ;host disk sector size  
31 0014 = hstprt    equ      20      ;host disk sectors/trk  
32 0004 = hstblk    equ      hstsz/128  
33 0050 = cpmspt    equ      hstblk * hstprt  
34 0003 = secnsk    equ      hstblk-1  
35          smask     hstblk  
36 0002 = secshf    equ      @x  
37          ;  
38          ;  
39          ;  
40          ;  
41          ;  
42          ;  
43 0000 = wrall    equ      0      ;write to allocated  
44 0001 = wrdir    equ      1      ;write to directory  
45 0002 = wrunal   equ      2      ;write to unallocated  
46          ;  
47          ;  
48          ;  
49          ;  
50          ;
```

the bdos entry points given below show the  
code which is relevant to deblocking only.

```

51
52
53
54      ; diskdef macro, or hand coded tables go here
55      dbase    equ    $                 ; disk Param block base
56
57 boot:
58 wboot:   ;enter here on system boot to initialize
59      0000  af          ;0 to accumulator
60      0001 326a01        ;host buffer inactive
61      0004 326c01        ;clear malloc count
62      0007 c9          ;i
63
64      ;home:
65      ;home the selected disk
66
67 home:   ;check for pending write
68      0008 3a6b01        ;lda
69      000b b7          ;ora
70      000c c21200        ;a
71      000f 326a01        ;hstdt
72      0012 c9          ;hstdt
73      0012 c9          ;ret
74      ;                                ;selected disk
75 seldisk: ;selected disk number
76      0013 79          ;mov  a,c
77      0014 326101        ;sta  seldisk
78      0017 6f          ;mov  l,a
79      0018 2600        ;mvi  h,0
80      ;                                ;disk number to hl
81      ;                                ;multiply by 16

```

```

82      dad    h
83      endm
84      001a+29   h
85      001b+29   h
86      001c+29   h
87      001d+29   h
88      001e 110000 lxi d,dPbase
89      0021 19    dad d
90      0022 c9    ret
91      ;
92      setrk:    ;set track given by registers bc
93          mov    h,b
94          0023 60
95          0024 69    mov    l,c
96          0025 225201 shld  seekrk
97          0028 c9    ret
98      ;
99      setsec:    ;set sector given by register C
100     0029 79    mov    a,c
101     002a 325401 sta    seeksec
102     002d c9    ret
103
104      ;
105      setdma:    ;set dma address given by bc
106     002e 60    mov    h,b
107     002f 69    mov    l,c
108     0030 227501 shld  dmaddr
109     0033 c9    ret
110

```

```

111
112      section:
113          ;translate sector number bc
114          0034 60      mov    hb
115          0035 69      mov    1,c
116          0036 c9      ret.
117
118
119
120          ; the read entry point takes the place of
121          ; the previous bios definition for read,
122
123
124      read:
125          ;read the selected cp/m sector
126          0037 af      xra   a
127          0038 326c01   sta    unacct
128          003b 3e01      mvi   a,1
129          003d 327301   sta    readop
130          0040 327201   sta    rsflag
131          0043 3e02      mvi   a,0
132          0045 327401   sta    wrtype
133          0048 c3bb00      jmp   TwoPer
134
135
136
137          ; the write entry point takes the place of
138          ; the previous bios definition for write,

```

130

140

141 ;

142 write:

143       ; iwrite the selected cff/m sector  
144        af           xra        a           ; to accumulator  
145        004c 327301     sta        readop   ; not a read operation  
146        004f 79        mov        a,c       ; write type in C  
147        0050 327401     sta        wrtypc   ; write unallocated?  
148        0053 fe02      cpi        wrtial   ; check for unalloc  
149        0050 c26f00     jnz        chkuna   ;

150        ; write to unallocated, set parameters  
151        0058 3d10      mvi        ahblksize/128   ; next unalloc recs  
152        005a 326c01     sta        unact    ; disk to seek  
153        005d 346101     lda        sekdisk   ; unadsk = sekdisk  
154        0060 326d01     sta        unadsk   ; unadsk = sectrk  
155        0063 246201     lhd        settrk   ; unatrk = sectrk  
156        0066 246e01     shld      unatrk   ; seksec = seksec  
157        0069 346401     lda        seksec   ; unasec = seksec  
158        006c 327001     sta        unasec   ;

159 ;  
160 chkuna:

161        ; check for write to unallocated sector  
162        006f 346c01     lda        unact   ; any unalloc remain?  
163        0072 b7        ora        a           ; skip if not  
164        0073 caae00     jz        alloc   ; same disk?  
165 ;  
166        ; more unallocated records remain  
167        0076 3d        dcr        a           ; unact = unact-1  
168        0077 346c01     sta        unact   ; same disk?  
169        007a 3a6101     lda        sekdisk

```

170 007d 216d01 lxi h,unadsk
171 0080 be cmp m
172 0081 c2ae00 jnz alloc
173 ; skip if not
174 ; disks are the same
175 0084 216e01 lxi h,unatrk
176 0087 cd5301 call sektrkcmp
177 008a c2ae00 jnz alloc
178 ; tracks are the same
179 008d 3a6401 lda seksec
180 0090 217001 lxi h,unasec
181 0093 be cmp m
182 0094 c2ae00 jnz alloc
183 ; skip if not
184 ; match, move to next sector for future ref
185 0097 34 int m
186 0098 7e mov a,m
187 0099 fe50 cpmpt
188 009b daa700 jc nouf
189 ; skip if no overflow
190 ; overflow to next track
191 009e 3600 movi m,o
192 009f 2a6e01 lhld unatrk
193 00a0 2a6e01 inx h
194 00a3 23 shld unatrk
195 00a4 226e01 ; nouf:
196 ; skip if no overflow
197 00a7 af xra a
198 00ab 327201 sta rsflag
199 ; mark as unnecessary read
200 ; to accumulator
; rsflag = 0

```

```

201    00ab c3b600 ;          jmp      rwoper   ;to perform the write
202
203 alloc:                                ;not an unallocated record, requires pre-read
204     00ae af          xra      a           ;0 to accum
205     00af 326601     sta      unach,    ;unach = 0
206     00b2 3c          inr      a           ;1 to accum
207     00b3 327701     sta      rsflag,   ;rsflag = 1
208
209
210
211 common code for read and write follows
212
213
214
215 rwoper:                                ;enter here to perform the read-write
216     00b6 af          xra      a           ;zero to accum
217     00b7 327101     sta      erflag,   ;no errors (yet)
218     00ba 3a6401     lda      seksec,   ;compute host sector
219
220     rept   secshft
221     ora      a           ;icarry = 0
222     ral
223     endm
224     ora      a           ;icarry = 0
225     lsl     rai,      ;shift right
226     ora      a           ;icarry = 0
227     ral
228     00c1 326601     sta      sekst,   ;host sector to seek

```

```

229      ; active host sector?          ; host active flag
230      00c4 216a01    lxi h,hsact
231      00c7 7e        mov a,m
232      00c8 3601    mvi m,1
233      00ca b7        ora a
234      00cb caf200   jz filhost
235      ; fill host if not
236      ;
237      ; host buffer active, same as seek buffer?
238      00ce 3a6101    lda sekdisk
239      00d1 216501    lxi h,hstdsk
240      00d4 be        cmp m
241      00d5 c2eb00   jnz nomatch
242      ;
243      ; same disk, same track?
244      00dB 216601    lxi h,hsttrk
245      00db cd5301    call sektrkcmp
246      00de c2eb00   jnz nomatch
247      ;
248      ; same disk, same track, same buffer?
249      00e1 3a6901    lda sekhost
250      00e4 216801    lxi h,hstsec
251      00e7 be        cmp m
252      00e8 ca0f01    jz match
253      ; skip if match
254      nomatch:
255      ; improper disk, but not correct sector
256      00eb 3a6b01    lda hstwrn
257      00ee b7        ora a
258      00ef c45f01    cmz writehst
259      ; clear host buff

```

```

259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289

; fillst:      ; may have to fill the host buffer
00f2 3a6101    lda     seeksk
00f5 326501    sta     hstdsk
00fb 2a6201    lhd    seekrk
00fb 226601    shld   hstrk
00fe 3a6901    lda     seekst
0101 326801    sta     hstsec
0104 3a7201    lda     rsflag
0107 b7        ora     a
0108 c46001    cnz     readnst
010b af        xra     a
010c 326b01    sta     hstart
;                         ; no pending write

; match:       ; copy data to or from buffer
010f 3a6d01    lda     seeksc
0112 e603    ani     secask
0114 6f        mou     1,a
0115 2600    mu1     h,0
;                         ; least signif bits
;                         ; ready to shift
;                         ; double count
;                         ; ishift left 7
0117+29    rep    7
0118+29    dad    h
0119+29    dad    h
011a+29    dad    h
011b+29    dad    h
011c+29    dad    h
011d+29    dad    h

```

```

290      ;          hl has relative host buffer address
291      011e 117701    lxi d,hostbuf
292      0121 19        dad d
293      0122 eb        xchg
294      0123 2a7501    lhld dmaadr
295      0126 0800      mvi c,i128
296      0128 3a7301    lda readop
297      012b b7        ora a
298      012c c23501    rmwne
299      ;          write operation, mark and switch direction
300      012f 3e01      mvi a,1
301      0131 326b01    sta hostwrt
302      0134 eb        xchg
303      ;          rwmwe:
304      0135 1a        ldx d
305      0136 13        inx d
306      0137 77        mov m,a
307      0138 23        inx h
308      0139 od        dcr c
309      013a c23501    rmwne
310      013b 19        lda wrtype
311      013c 13        wrdr
312      013d 3a7401    cpi wrdr
313      013e 19        lda errflag
314      013f 13        rmwz
315      0140 fe01      ldi directory?
316      0142 3a7101    jne error
317      0145 c0        jne further_processing

```

319       ; clear host buffer for directory write  
320        0146 b7       ora     a  
321        ; errors?  
322        0147 c0       rmz  
323        ; skip if 50  
324        0148 af       xra    a  
325        0149 326b01    sta     hstwrt  
326        014c cd5f01   call    writehst  
327        014f 3a7101   lda     erflag  
328        0152 c9       ret

329        ; utility subroutine for 16-bit compare  
330        ;  
331        ;  
332        ;  
333        ;

sektrkcmp:

334        ihl = ,unatrk or ,hstrk, compare with sektrk  
335        0153 eb       xchg  
336        ; compare with sektrk  
337        0154 216201    lxr    h,sektrk  
338        0157 1a       ldax    d  
339        0158 be       cmp     m  
340        0159 c0       rmz  
341        ; return if not  
342        015a 13       inx    d  
343        015b 23       imx    h  
344        015c 1a       ldax    d  
345        015d be       cmp     m  
346        015e c9       sets flags  
          ret

```
347
348
349      writehst performs the physical write to
350      the host disk, readhst reads the physical
351      disk.
352
353
354
355      writehst:
356          ihstdsk = host disk #, hstrk = host track #,
357          ihstsec = host sect #, write "hstiz" bytes
358          if from hstbuf and return error flag in erflag,
359          ireturn erflag non-zero if error
360          ret.
361
362      readhst:
363          ihstdsk = host disk #, hstrk = host track #,
364          ihstsec = host sect #, read "hstsiz" bytes
365          into hstbuf and return error flag in erflag,
366          ret
367
368
369
370
371
372
373
374      sekdisk:    ds      1      ;seek disk number
375      sektrk:    ds      2      ;seek track number
376      seksec:    ds      1      ;seek sector number
```

```

377      i      hstdsk:    ds     1      ;host disk number
378      0165    hstrk:    ds     2      ;host track number
379      0166    hssec:    ds     1      ;host sector number
380      0168    i
381          hshst:    ds     1      ;seek shr secshf
382      0169    hsact:    ds     1      ;host active flag
383      016a    hstwrt:   ds     1      ;host written flag
384      016b    i
385          i
386      016c    unaent:   ds     1      ;unalloc rec cnt
387      016d    unadsk:   ds     1      ;last unalloc disk
388      016e    unatrk:   ds     2      ;last unalloc track
389      0170    unasec:   ds     1      ;last unalloc sector
390          i
391      0171    erflag:   ds     1      ;error reporting
392      0172    rsflag:   ds     1      ;read sector flag
393      0173    readop:   ds     1      ;if read operation
394      0174    wrtype:   ds     1      ;write operation type
395      0175    dmaadr:   ds     2      ;last dma address
396      0177    hstbuf:   ds     hstsiz ;host buffer
397          i
398          i
399          i      ;the endef macro invocation goes here
400          i
401          i
402          i
403      0377    end

```

alloc	164	172	177	183	203#
blksize	0800	29#	151		
boot	0000	57#			
chkuna	006f	148	160#		
cPMSPT	0050	33*	188		
dmaadr	0175	109	294	395#	
database	0000	55#	88		
erflag	0171	218	317	326	391#
filist	00f2	235	260#		
home	0008	65#	67#		
homel	0012	70	72#		
hstact	016a	61	71	231	383#
hstlk	0004	32*	33	34	35
hstluf	0177	291	396#		
hstdsk	0165	239	263	378#	
hstsec	0168	250	267	380#	
hstiz	0200	30#	32	395	
hstpT	0014	31#	33		
hstrk	0166	244	265	379#	
hsturt	016b	68	256	272	302
match	010f1	252	274#		
nomatch	00eb	241	246	254#	
noof	00a7	189	197#		
read	0037	124#			
readhst	0160	270	362#		
readop	0173	129	144	296	393#
rsflag	0172	130	200	208	392#
rwmive	0135	298	305#	312	
rwoper	00b6	133	211	215#	
secksk	0003	34#	277		
sechsf	0002	36#	220		

sectoran

0034 112#  
sekdisk 0161 78  
sekfst 0169 228  
seksec 0164 102  
sektrk 0162 96  
sektrkcmp 0153 176  
seldisk 0013 75#  
setdma 002e 105#  
setsec 0029 99#  
settrk 0023 92#  
unacct 016c 62  
unadsk 016d 154  
unadsk 0170 158  
unasec 016e 156  
unatrk 016f 175  
wboot 0000 58#  
wraill 0000 43#  
wrdir 0001 44#  
write 004b 141#  
writehst 015f 258  
wrttyp 0174 132  
wrual 0002 45#

238 262 374#

266 382#

157 180 219 276 376#

155 264 337 375#

245 334#

105#

99#

92#

127 152 162 168 206 386#

170 387#

181 389#

193 195 388#

43#

316

141#

325 355#

146 315 394#

131 147

# Appendix H

## Glossary

**address:** Number representing the location of a byte in memory. Within CP/M there are two kinds of addresses: logical and physical. A physical address refers to an absolute and unique location within the computer's memory space. A logical address refers to the offset or displacement of a byte in relation to a base location. A standard CP/M program is loaded at address 0100H, the base value; the first instruction of a program has a physical address of 0100H and a relative address or offset of OH.

**allocation vector (ALV):** An allocation vector is maintained in the BIOS for each logged-in disk drive. A vector consists of a string of bits, one for each block on the drive. The bit corresponding to a particular block is set to one when the block has been allocated and to zero otherwise. The first two bytes of this vector are initialized with the bytes AL0 and AL1 on, thus allocating the directory blocks. CP/M Function 27 returns the allocation vector address.

**AL0, AL1:** Two bytes in the disk parameter block that reserve data blocks for the directory. These two bytes are copied into the first two bytes of the allocation vector when a drive is logged in. See **allocation vector**.

**ALV:** See **allocation vector**.

**ambiguous filename:** Filename that contains either of the CP/M wildcard characters, ? or \*, in the primary filename, filetype, or both. When you replace characters in a filename with these wildcard characters, you create an ambiguous filename and can easily reference more than one CP/M file in a single command line.

**American Standard Code for Information Interchange:** See **ASCII**.

**applications program:** Program designed to solve a specific problem. Typical applications programs are business accounting packages, word processing (editing) programs and mailing list programs.

**archive attribute:** File attribute controlled by the high-order bit of the t3 byte (FCB + 11) in a directory element. This attribute is set if the file has been archived.

**argument:** Symbol, usually a letter, indicating a place into which you can substitute a number, letter, or name to give an appropriate meaning to the formula in question.

**ASCII:** American Standard Code for Information Interchange. ASCII is a standard set of seven-bit numeric character codes used to represent characters in memory. Each character requires one byte of memory with the high-order bit usually set to zero. Characters can be numbers, letters, and symbols. An ASCII file can be intelligibly displayed on the video screen or printed on paper.

**assembler:** Program that translates assembly language into the binary machine code. Assembly language is simply a set of mnemonics used to designate the instruction set of the CPU. See ASM in Section 3 of this manual.

**back-up:** Copy of a disk or file made for safekeeping, or the creation of the duplicate disk or file.

**Basic Disk Operating System:** See BDOS.

**BDOS:** Basic Disk Operating System. The BDOS module of the CP/M operating system provides an interface for a user program to the operating system. This interface is in the form of a set of function calls which may be made to the BDOS through calls to location 0005H in page zero. The user program specifies the number of the desired function in register C. User programs running under CP/M should use BDOS functions for all I/O operations to remain compatible with other CP/M systems and future releases. The BDOS normally resides in high memory directly below the BIOS.

**bias:** Address value which when added to the origin address of your BIOS module produces 1F80H, the address of the BIOS module in the MOVCPM image. There is also a bias value that when added to the BOOT module origin produces 0900H, the address of the BOOT module in the MOVCPM image. You must use these bias values with the R command under DDT or SID™ when you patch a CP/M system. If you do not, the patched system may fail to function.

**binary:** Base 2 numbering system. A binary digit can have one of two values: 0 or 1. Binary numbers are used in computers because the hardware can most easily exhibit two states: off and on. Generally, a bit in memory represents one binary digit.

**Basic Input/Output System:** See BIOS.

**BIOS:** Basic Input/Output System. The BIOS is the only hardware-dependent module of the CP/M system. It provides the BDOS with a set of primitive I/O operations. The BIOS is an assembly language module usually written by the user, hardware manufacturer, or independent software vendor, and is the key to CP/M's portability. The BIOS

interfaces the CP/M system to its hardware environment through a standardized jump table at the front of the BIOS routine and through a set of disk parameter tables which define the disk environment. Thus, the BIOS provides CP/M with a completely table-driven I/O system.

**BIOS base:** Lowest address of the BIOS module in memory, that by definition must be the first entry point in the BIOS jump table.

**bit:** Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes, eight bits to a byte, which is the smallest directly addressable unit in an Intel 8080 or Zilog Z80. By common convention, the bits in a byte are numbered from right, 0 for the low-order bit, to left, 7 for the high-order bit. Bit values are often represented in hexadecimal notation by grouping the bits from the low-order bit in groups of four. Each group of four bits can have a value from 0 to 15 and thus can easily be represented by one hexadecimal digit.

**BLM:** See block mask.

**block:** Basic unit of disk space allocation. Each disk drive has a fixed block size (BLS) defined in its disk parameter block in the BIOS. A block can consist of 1K, 2K, 4K, 8K, or 16K consecutive bytes. Blocks are numbered relative to zero so that each block is unique and has a byte displacement in a file equal to the block number times the block size.

**block mask (BLM):** Byte value in the disk parameter block at DPB + 3. The block mask is always one less than the number of 128 byte sectors that are in one block. Note that  $BLM = (2^{**} BSH) - 1$ .

**block shift (BSH):** Byte parameter in the disk parameter block at DPB + 2. Block shift and block mask (BLM) values are determined by the block size (BLS). Note that  $BLM = (2^{**} BSH) - 1$ .

**blocking & deblocking algorithm:** In some disk subsystems the disk sector size is larger than 128 bytes, usually 256, 512, 1024, or 2048 bytes. When the host sector size is larger than 128 bytes, host sectors must be buffered in memory and the 128-byte CP/M sectors must be blocked and deblocked by adding an additional module, the blocking and deblocking algorithm, between the BIOS disk I/O routines and the actual disk I/O. The host sector size must be an even multiple of 128 bytes for the algorithm to work correctly. The blocking and deblocking algorithm allows the BDOS and BIOS to function exactly as if the entire disk consisted only of 128-byte sectors, as in the standard CP/M installation.

**BLS:** Block size in bytes. See **block**.

**boot:** Process of loading an operating system into memory. A boot program is a small piece of code that is automatically executed when you power-up or reset your computer. The boot program loads the rest of the operating system into memory in a manner similar to a person pulling himself up by his own bootstraps. This process is sometimes called a cold boot or cold start. Bootstrap procedures vary from system to system. The boot program must be customized for the memory size and hardware environment that the operating system manages. Typically, the boot resides on the first sector of the system tracks on your system disk. When executed, the boot loads the remaining sectors of the system tracks into high memory at the location for which the CP/M system has been configured. Finally, the boot transfers execution to the boot entry point in the BIOS jump table so that the system can initialize itself. In this case, the boot program should be placed at 900H in the SYSGEN image. Alternatively, the boot program may be located in ROM.

**bootstrap:** See **boot**.

**BSH:** See **block shift**.

**BTREE:** General purpose file access method that has become the standard organization for indexes in large data base systems. BTREE provides near optimum performance over the full range of file operations, such as insertion, deletion, search, and search next.

**buffer:** Area of memory that temporarily stores data during the transfer of information.

**built-in commands:** Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

**byte:** Unit of memory or disk storage containing eight bits. A byte can represent a binary number between 0 and 255, and is the smallest unit of memory that can be addressed directly in 8-bit CPUs such as the Intel 8080 or Zilog Z80.

**CCP:** Console Command Processor. The CCP is a module of the CP/M operating system. It is loaded directly below the BDOS module and interprets and executes commands typed by the console user. Usually these commands are programs that the CCP loads and calls. Upon completion, a command program may return control to the CCP if it has not overwritten it. If it has, the program can reload the CCP into memory

by a warm boot operation initiated by either a jump to zero, BDOS system reset (Function 0), or a cold boot. Except for its location in high memory, the CCP works like any other standard CP/M program; that is, it makes only BDOS function calls for its I/O operations.

**CCP base:** Lowest address of the CCP module in memory. This term sometimes refers to the base of the CP/M system in memory, as the CCP is normally the lowest CP/M module in high memory.

**checksum vector (CSV):** Contiguous data area in the BIOS, with one byte for each directory sector to be checked, that is, CKS bytes. See **CKS**. A checksum vector is initialized and maintained for each logged-in drive. Each directory access by the system results in a checksum calculation that is compared with the one in the checksum vector. If there is a discrepancy, the drive is set to Read-Only status. This feature prevents the user from inadvertently switching disks without logging in the new disk. If the new disk is not logged-in, it is treated the same as the old one, and data on it might be destroyed if writing is done.

**CKS:** Number of directory records to be checked summed on directory accesses. This is a parameter in the disk parameter block located in the BIOS. If the value of CKS is zero, then no directory records are checked. CKS is also a parameter in the diskdef macro library, where it is the actual number of directory elements to be checked rather than the number of directory records.

**cold boot:** See **boot**. Cold boot also refers to a jump to the boot entry point in the BIOS jump table.

**COM:** Filetype for a CP/M command file. See **command file**.

**command:** CP/M command line. In general, a CP/M command line has three parts: the command keyword, command tail, and a carriage return. To execute a command, enter a CP/M command line directly after the CP/M prompt at the console and press the carriage return or enter key.

**command file:** Executable program file of filetype COM. A command file is a machine language object module ready to be loaded and executed at the absolute address of 0100H. To execute a command file, enter its primary filename as the command keyword in a CP/M command line.

**command keyword:** Name that identifies a CP/M command, usually the primary filename of a file of type COM, or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

**command syntax:** Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

**command tail:** Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and filetype, and options or parameters. Some commands do not require a command tail.

**CON:** Mnemonic that represents the CP/M console device. For example, the CP/M command PIP CON:=TEST.SUB displays the file TEST.SUB on the console device. The explanation of the STAT command tells how to assign the logical device CON: to various physical devices. See **console**.

**concatenate:** Name of the PIP operation that copies two or more separate files into one new file in the specified sequence.

**concurrency:** Execution of two processes or operations simultaneously.

**CONIN:** BIOS entry point to a routine that reads a character from the console device.

**CONOUT:** BIOS entry point to a routine that sends a character to the console device.

**console:** Primary input/output device. The console consists of a listing device, such as a screen or teletype, and a keyboard through which the user communicates with the operating system or applications program.

**Console Command Processor:** See CCP.

**CONST:** BIOS entry point to a routine that returns the status of the console device.

**control character:** Nonprinting character combination. CP/M interprets some control characters as simple commands such as line editing functions. To enter a control character, hold down the CONTROL key and strike the specified character key.

**Control Program for Microcomputers:** See CP/M.

**CP/M:** Control Program for Microcomputers. An operating system that manages computer resources and provides a standard systems interface to software written for a large variety of microprocessor-based computer systems.

**CP/M 1.4 compatibility:** For a CP/M 2 system to be able to read correctly single-density disks produced under a CP/M 1.4 system, the extent mask must be zero and the block size 1K. This is because under CP/M 2 an FCB may contain more than one extent. The number of extents that may be contained by an FCB is EXM + 1. The issue of CP/M 1.4 compatibility also concerns random file I/O. To perform random file I/O under CP/M 1.4, you must maintain an FCB for each extent of the file. This scheme is upward compatible with CP/M 2 for files not exceeding 512K bytes, the largest file size supported under CP/M 1.4. If you wish to implement random I/O for files larger than 512K bytes under CP/M 2, you must use the random read and random write functions, BDOS functions 33, 34, and 36. In this case, only one FCB is used, and if CP/M 1.4 compatibility is required, the program must use the return version number function, BDOS Function 12, to determine which method to employ.

**CP/M prompt:** Characters that indicate that CP/M is ready to execute your next command. The CP/M prompt consists of an upper-case letter, A-P, followed by a > character; for example, A>. The letter designates which drive is currently logged in as the default drive. CP/M will search this drive for the command file specified, unless the command is a built-in command or prefaced by a select drive command: for example, B:STAT.

**CP/NET:** Digital Research network operating system enabling microcomputers to obtain access to common resources via a network. CP/NET consists of MP/M masters and CP/M slaves with a network interface between them.

**CSV:** See **checksum vector**.

**cursor:** One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

**data file:** File containing information that will be processed by a program.

**deblocking:** See **blocking & deblocking algorithm**.

**default:** Currently selected disk drive and user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M is first invoked, the default disk drive is drive A, and the default user number is 0.

**default buffer:** Default 128-byte buffer maintained at 0080H in page zero. When the CCP loads a COM file, this buffer is initialized to the command tail; that is, any characters typed after the COM file name are loaded into the buffer. The first byte at 0080H contains the length of the command tail, while the command tail itself begins at 0081H. The command tail is terminated by a byte containing a binary zero value. The I command under DDT and SID initializes this buffer in the same way as the CCP.

**default FCB:** Two default FCBs are maintained by the CCP at 005CH and 006CH in page zero. The first default FCB is initialized from the first delimited field in the command tail. The second default FCB is initialized from the next field in the command tail.

**delimiter:** Special characters that separate different items in a command line; for example, a colon separates the drive specification from the filename. The CCP recognizes the following characters as delimiters: . : = ; < > \_\_, blank, and carriage return. Several CP/M commands also treat the following as delimiter characters: , [ ] ( ) \$. It is advisable to avoid the use of delimiter characters and lower-case characters in CP/M filenames.

**DIR:** Parameter in the diskdef macro library that specifies the number of directory elements on the drive.

**DIR attribute:** File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number and drive only.

**DIRBUF:** 128-byte scratchpad area for directory operations, usually located at the end of the BIOS. DIRBUF is used by the BDOS during its directory operations. DIRBUF also refers to the two-byte address of this scratchpad buffer in the disk parameter header at DPbase + 8 bytes.

**directory:** Portion of a disk that contains entries for each file on the disk. In response to the DIR command, CP/M displays the filenames stored in the directory. The directory also contains the locations of the blocks allocated to the files. Each file directory element is in the form of a 32-byte FCB, although one file can have several elements, depending on its size. The maximum number of directory elements supported is specified by the drive's disk parameter block value for DRM.

**directory element:** Data structure. Each file on a disk has one or more 32-byte directory elements associated with it. There are four directory elements per directory sector. Directory elements can also be referred to as directory FCBs.

**directory entry:** File entry displayed by the DIR command. Sometimes this term refers to a physical directory element.

**disk, diskette:** Magnetic media used for mass storage in a computer system. Programs and data are recorded on the disk in the same way music can be recorded on cassette tape. The CP/M operating system must be initially loaded from disk when the computer is turned on. Diskette refers to smaller capacity removable floppy diskettes, while disk may refer to either a diskette, removable cartridge disk, or fixed hard disk. Hard disk capacities range from five to several hundred megabytes of storage.

**diskdef macro library:** Library of code that when used with MAC, the Digital Research macro assembler, creates disk definition tables such as the DPB and DPH automatically.

**disk drive:** Peripheral device that reads and writes information on disk. CP/M assigns a letter to each drive under its control. For example, CP/M may refer to the drives in a four-drive system as A, B, C, and D.

**disk parameter block (DPB):** Data structure referenced by one or more disk parameter headers. The disk parameter block defines disk characteristics in the fields listed below:

SPT is the total number of sectors per track.

BSH is the data allocation block shift factor.

BLM is the data allocation block mask.

EXM is the extent mask determined by BLS and DSM.

DSM is the maximum data block number.

DRM is the maximum number of directory entries—1.

AL0 reserves directory blocks.

AL1 reserves directory blocks.

CKS is the number of directory sectors check summed.

OFF is the number of reserved system tracks.

The address of the disk parameter block is located in the disk parameter header at DPbase +0AH. CP/M Function 31 returns the DPB address. Drives with the same characteristics can use the same disk parameter header, and thus the same DPB. However, drives with different characteristics must each have their own disk parameter header and disk parameter blocks. When the BDOS calls the SELDSK entry point in the BIOS, SELDSK must return the address of the drive's disk parameter header in register HL.

**disk parameter header (DPH):** Data structure that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The disk parameter header contains six bytes of scratchpad area for the BDOS, and the following five 2-byte parameters:

XLT is the sector translation table address.

DIRBUF is the directory buffer address.

DPB is the disk parameter block address.

CSV is the checksum vector address.

ALV is the allocation vector address.

Given n disk drives, the disk parameter headers are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n - 1.

**DKS:** Parameter in the diskdef macro library specifying the number of data blocks on the drive.

**DMA:** Direct Memory Access. DMA is a method of transferring data from the disk into memory directly. In a CP/M system, the BDOS calls the BIOS entry point READ to read a sector from the disk into the currently selected DMA address. The DMA address must be the address of a 128-byte buffer in memory, either the default buffer at 0080H in page zero, or a user-assigned buffer in the TPA. Similarly, the BDOS calls the BIOS entry point WRITE to write the record at the current DMA address to the disk.

**DN:** Parameter in the diskdef macro library specifying the logical drive number.

**DPB:** See **disk parameter block**.

**DPH:** See **disk parameter header**.

**DRM:** 2-byte parameter in the disk parameter block at DPB + 7. DRM is one less than the total number of directory entries allowed for the drive. This value is related to DPB bytes AL0 and AL1, which allocates up to 16 blocks for directory entries.

**DSM:** 2-byte parameter of the disk parameter block at DPB + 5. DSM is the maximum data block number supported by the drive. The product BLS times (DSM + 1) is the total number of bytes held by the drive. This must not exceed the capacity of the physical disk less the reserved system tracks.

**editor:** Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The CP/M editor is invoked by typing the command ED next to the system prompt on the console.

**EX:** Extent number field in an FCB. See **extent**.

**executable:** Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot execute names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

**execute a program:** Start the processing of executable code.

**EXM:** See **extent mask**.

**extent:** 16K consecutive bytes in a file. Extents are numbered from 0 to 31. One extent can contain 1, 2, 4, 8, or 16 blocks. EX is the extent number field of an FCB and is a one-byte field at FCB + 12, where FCB labels the first byte in the FCB. Depending on the block size (BLS) and the maximum data block number (DSM), an FCB can contain 1, 2, 4, 8, or 16 extents. The EX field is normally set to 0 by the user but contains the current extent number during file I/O. The term FCB folding describes FCBs containing more than one extent. In CP/M version 1.4, each FCB contained only one extent. Users attempting to perform random record I/O and maintain CP/M 1.4 compatibility should be aware of the implications of this difference. See **CP/M 1.4 compatibility**.

**extent mask (EXM):** A byte parameter in the disk parameter block located at DPB + 3. The value of EXM is determined by the block size (BLS) and whether the maximum data block number (DSM) exceeds 255. There are EXM + 1 extents per directory FCB.

**FCB:** See **File Control Block**.

**file:** Collection of characters, instructions, or data that can be referenced by a unique identifier. Files are usually stored on various types of media, such as disk, or magnetic tape. A CP/M file is identified by a file specification and resides on disk as a collection of from zero to 65,536 records. Each record is 128 bytes and can contain either binary or ASCII data. Binary files contain bytes of data that can vary in value from 0H to 0FFH. ASCII files contain sequences of character codes delineated by a carriage return and line-feed combination; normally byte values range from 0H to 7FH. The directory maps the file as a series of physical blocks. Although files are defined as a sequence of consecutive logical records, these records can not reside in consecutive sectors on the disk. See also **block**, **directory**, **extent**, **record**, and **sector**.

**File Control Block (FCB):** Structure used for accessing files on disk. Contains the drive, filename, filetype, and other information describing a file to be accessed or created on the disk. A file control block consists of 36 consecutive bytes specified by the user for file I/O functions. FCB can also refer to a directory element in the directory portion of the allocated disk space. These contain the same first 32 bytes of the FCB, but lack the current record and random record number bytes.

**filename:** Name assigned to a file. A filename can include a primary filename of one to eight characters; a filetype of zero to three characters. A period separates the primary filename from the filetype.

**file specification:** Unique file identifier. A complete CP/M file specification includes a disk drive specification followed by a colon, d:, a primary filename of one to eight characters, a period, and a filetype of zero to three characters. For example, b:example.tex is a complete CP/M file specification.

**filetype:** Extension to a filename. A filetype can be from zero to three characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Some programs require that files to be processed have specific filenames.

**floppy disk:** Flexible magnetic disk used to store information. Floppy disks come in 5 1/4- and 8-inch diameters.

**FSC:** Parameter in the diskdef macro library specifying the first physical sector number. This parameter is used to determine SPT and build XLT.

**hard disk:** Rigid, platter-like, magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

**hardware:** Physical components of a computer.

**hexadecimal notation:** Notation for base 16 values using the decimal digits and letters A, B, C, D, E, and F to represent the 16 digits. Hexadecimal notation is often used to refer to binary numbers. A binary number can be easily expressed as a hexadecimal value by taking the bits in groups of 4, starting with the least significant bit, and expressing each group as a hexadecimal digit, 0-F. Thus the bit value 1011 becomes 0BH and 10110101 becomes 0B5H.

**hex file:** ASCII-printable representation of a command, machine language, file.

**hex file format:** Absolute output of ASM and MAC for the Intel 8080 is a hex format file, containing a sequence of absolute records that give a load address and byte values to be stored, starting at the load address.

**HOME:** BIOS entry point which sets the disk head of the currently selected drive to the track zero position.

**host:** Physical characteristics of a hard disk drive in a system using the blocking and deblocking algorithm. The term, host, helps distinguish physical hardware characteristics from CP/M's logical characteristics. For example, CP/M sectors are always 128 bytes, although the host sector size can be a multiple of 128 bytes.

**input:** Data going into the computer, usually from an operator typing at the terminal or by a program reading from the disk.

**input/output:** See I/O.

**interface:** Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

**I/O:** Abbreviation for input/output. Usually refers to input/output operations or routines handling the input and output of data in the computer system.

**IOBYTE:** A one-byte field in page zero, currently at location 0003H, that can support a logical-to-physical device mapping for I/O. However, its implementation in your BIOS is purely optional and might or might not be supported in a given CP/M system. The IOBYTE is easily set using the command:

STAT <logical device> = <physical device>

The CP/M logical devices are CON:, RDR:, PUN:, and LST:; each of these can be assigned to one of four physical devices. The IOBYTE can be initialized by the BOOT entry point of the BIOS and interpreted by the BIOS I/O entry points CONST, CONIN, CONOUT, LIST, PUNCH, and READER. Depending on the setting of the IOBYTE, different I/O drivers can be selected by the BIOS. For example, setting LST:=TTY: might cause LIST output to be directed to a serial port, while setting LST:=LPT: causes LIST output to be directed to a parallel port.

**K:** Abbreviation for kilobyte. See kilobyte.

**keyword:** See **command keyword**.

**kilobyte (K):** 1024 bytes or 0400H bytes of memory. This is a standard unit of memory. For example, the Intel 8080 supports up to 64K of memory address space or 65,536 bytes. 1024 kilobytes equal one megabyte, or over one million bytes.

**linker:** Utility program used to combine relocatable object modules into an absolute file ready for execution. For example, LINK-80™ creates either a COM or PRL file from relocatable REL files, such as those produced by PL/I-80™.

**LIST:** A BIOS entry point to a routine that sends a character to the list device, usually a printer.

**list device:** Device such as a printer onto which data can be listed or printed.

**LISTST:** BIOS entry point to a routine that returns the ready status of the list device.

**loader:** Utility program that brings an absolute program image into memory ready for execution under the operating system, or a utility used to make such an image. For example, LOAD prepares an absolute COM file from the assembler hex file output that is ready to be executed under CP/M.

**logged in:** Made known to the operating system, in reference to drives. A drive is logged in when it is selected by the user or an executing process. It remains selected or logged in until you change disks in a floppy disk drive or enter CTRL-C at the command level, or until a BDOS Function 0 is executed.

**logical:** Representation of something that might or might not be the same in its actual physical form. For example, a hard disk can occupy one physical drive, yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

**logical sector:** See **sector**.

**logical-to-physical sector translation table:** See **XLT**.

**LSC:** Diskdef macro library parameter specifying the last physical sector number.

**LST:** Logical CP/M list device, usually a printer. The CP/M list device is an output-only device referenced through the LIST and LISTST entry points of the BIOS. The STAT command allows assignment of LST: to one of the physical devices: TTY:, CRT:, LPT:, or UL1:, provided these devices and the IOBYTE are implemented in the LIST and LISTST entry points of your CP/M BIOS module. The CP/NET command NETWORK allows assignment of LST: to a list device on a network master. For example, PIP LST:=TEST.SUB prints the file TEST.SUB on the list device.

**macro assembler:** Assembler code translator providing macro processing facilities. Macro definitions allow groups of instructions to be stored and substituted in the source program as the macro names are encountered. Definitions and invocations can be nested and macro parameters can be formed to pass arbitrary strings of text to a specific macro for substitution during expansion.

**megabyte:** Over one million bytes; 1024 kilobytes. See **byte**, and **kilobyte**.

**microprocessor:** Silicon chip that is the central processing unit (CPU) of the microcomputer. The Intel 8080 and the Zilog Z80 are microprocessors commonly used in CP/M systems.

**MOVCPM image:** Memory image of the CP/M system created by MOVCPM. This image can be saved as a disk file using the SAVE command or placed on the system tracks using the SYSGEN command without specifying a source drive. This image varies, depending on the presence of a one-sector or two-sector boot. If the boot is less than 128 bytes (one sector), the boot begins at 0900H, the CP/M system at 0980H, and the BIOS at 1F80H. Otherwise, the boot is at 0900H, the CP/M system at 1000H, and the BIOS at 2000H. In a CP/M 1.4 system with a one-sector boot, the addresses are the same as for the CP/M 2 system—except that the BIOS begins at 1E80H instead of 1F80H.

**MP/M:** Multi-Programming Monitor control program. A microcomputer operating system supporting multi-terminal access with multi-programming at each terminal.

**multi-programming:** The capability of initiating and executing more than one program at a time. These programs, usually called processes, are time-shared, each receiving a slice of CPU time on a round-robin basis. See **concurrency**.

**nibble:** One half of a byte, usually the high-order or low-order 4 bits in a byte.

**OFF:** Two-byte parameter in the disk parameter block at DPB + 13 bytes. This value specifies the number of reserved system tracks. The disk directory begins in the first sector of track OFF.

**OFS:** Diskdef macro library parameter specifying the number of reserved system tracks. See **OFF**.

**operating system:** Collection of programs that supervises the execution of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices. It enables user-written programs to execute safely. An operating system standardizes the use of computer resources for the programs running under it.

**option:** One of many parameters that can be part of a command tail. Use options to specify additional conditions for a command's execution.

**output:** Data that is sent to the console, disk, or printer.

**page:** 256 consecutive bytes in memory beginning on a page boundary, whose base address is a multiple of 256 (100H) bytes. In hex notation, pages always begin at an address with a least significant byte of zero.

**page relocatable program:** See **PRL**.

**page zero:** Memory region between 0000H and 0100H used to hold critical system parameters. Page zero functions primarily as an interface region between user programs and the CP/M BDOS module. Note that in non-standard systems this region is the base page of the system and represents the first 256 bytes of memory used by the CP/M system and user programs running under it.

**parameter:** Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

**peripheral devices:** Devices external to the CPU. For example, terminals, printers, and disk drives are common peripheral devices that are not part of the processor but are used in conjunction with it.

**physical:** Characteristic of computer components, generally hardware, that actually exist. In programs, physical components can be represented by logical components.

**primary filename:** First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains one to eight characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

**PRL:** Page relocatable program. A page relocatable program is stored on disk with a PRL filetype. Page relocatable programs are easily relocated to any page boundary and thus are suitable for execution in a nonbanked MP/M system.

**program:** Series of coded instructions that performs specific tasks when executed by a computer. A program can be written in a processor-specific language or a high-level language that can be implemented on a number of different processors.

**prompt:** Any characters displayed on the video screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The alphabetic character indicates the default drive. Some applications programs have their own special prompts. See **CP/M prompt**.

**PUN:** Logical CP/M punch device. The punch device is an output-only device accessed through the PUNCH entry point of the BIOS. In certain implementations, PUN: can be a serial device such as a modem.

**PUNCH:** BIOS entry point to a routine that sends a character to the punch device.

**RDR:** Logical CP/M reader device. The reader device is an input-only device accessed through the READER entry point in the BIOS. See **PUN:**.

**READ:** Entry point in the BIOS to a routine that reads 128 bytes from the currently selected drive, track, and sector into the current DMA address.

**READER:** Entry point to a routine in the BIOS that reads the next character from the currently assigned reader device.

**Read-Only (R/O):** Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the Read-Only attribute allows you to read from that file but not write to it. When assigned to a drive, the Read-Only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The STAT command can set a file or a drive to Read-Only. Every file and drive is either Read-Only or Read-Write. The default setting for drives and files is Read-Write, but an error in resetting the disk or changing media automatically sets the drive to Read-Only until the error is corrected. See also **ROM**.

**Read-Write (R/W):** Attribute that can be assigned to a disk file or a disk drive. The Read-Write attribute allows you to read from and write to a specific Read-Write file or to any file on a disk that is in a drive set to Read-Write. A file or drive can be set to either Read-Only or Read-Write.

**record:** Group of bytes in a file. A physical record consists of 128 bytes and is the basic unit of data transfer between the operating system and the application program. A logical record might vary in length and is used to represent a unit of information. Two 64-byte employee records can be stored in one 128-byte physical record. Records are grouped together to form a file.

**recursive procedure:** Code that can call itself during execution.

**reentrant procedure:** Code that can be called by one process while another is already executing it. Thus, reentrant code can be shared between different users. Reentrant procedures must not be self-modifying; that is, they must be pure code and not contain data. The data for reentrant procedures can be kept in a separate data area or placed on the stack.

**restart (RST):** One-byte call instruction usually used during interrupt sequences and for debugger break pointing. There are eight restart locations, RST 0 through RST 7, whose addresses are given by the product of 8 times the restart number.

**R/O:** See **Read-Only**.

**ROM:** Read-Only memory. This memory can be read but not written and so is suitable for code and preinitialized data areas only.

**RST:** See **restart**.

**R/W:** See **Read-Write**.

**sector:** In a CP/M system, a sector is always 128 consecutive bytes. A sector is the basic unit of data read and written on the disk by the BIOS. A sector can be one 128-byte record in a file or a sector of the directory. The BDOS always requests a logical sector number between 0 and (SPT - 1). This is typically translated into a physical sector by the BIOS entry point SECTRAN. In some disk subsystems, the disk sector size is larger than 128 bytes, usually a power of two, such as 256, 512, 1024, or 2048 bytes. These disk sectors are always referred to as host sectors in CP/M documentation and should not be confused with other references to sectors, in which cases the CP/M 128-byte sectors should be assumed. When the host sector size is larger than 128 bytes, host sectors must be buffered in memory and the 128-byte CP/M sectors must be blocked and deblocked from them. This can be done by adding an additional module, the blocking and deblocking algorithm, between the BIOS disk I/O routines and the actual disk I/O.

**sectors per track (SPT):** A two-byte parameter in the disk parameter block at DPB + 0. The BDOS makes calls to the BIOS entry point SECTRAN with logical sector numbers ranging between 0 and (SPT - 1) in register BC.

**SECTRAN:** Entry point to a routine in the BIOS that performs logical-to-physical sector translation for the BDOS.

**SELDSK:** Entry point to a routine in the BIOS that sets the currently selected drive.

**SETDMA:** Entry point to a routine in the BIOS that sets the currently selected DMA address. The DMA address is the address of a 128-byte buffer region in memory that is used to transfer data to and from the disk in subsequent reads and writes.

**SETSEC:** Entry point to a routine in the BIOS that sets the currently selected sector.

**SETTRK:** Entry point to a routine in the BIOS that sets the currently selected track.

**skew factor:** Factor that defines the logical-to-physical sector number translation in XLT. Logical sector numbers are used by the BDOS and range between 0 and (SPT - 1). Data is written in consecutive logical 128-byte sectors grouped in data blocks. The number of sectors per block is given by BLS/128. Physical sectors on the disk media are also numbered consecutively. If the physical sector size is also 128 bytes, a one-to-one relationship exists between logical and physical sectors. The logical-to-physical translation table (XLT) maps this relationship, and a skew factor is typically used in generating the table entries. For instance, if the skew factor is 6, XLT will be:

Logical:	0	1	2	3	4	5	6	...	25
Physical:	1	7	13	19	25	5	11	...	22

The skew factor allows time for program processing without missing the next sector. Otherwise, the system must wait for an entire disk revolution before reading the next logical sector. The skew factor can be varied, depending on hardware speed and application processing overhead. Note that no sector translation is done when the physical sectors are larger than 128 bytes, as sector deblocking is done in this case. See also **sector**, **SKF**, and **XLT**.

**SKF:** A diskdef macro library parameter specifying the skew factor to be used in building XLT. If SKF is zero, no translation table is generated and the XLT byte in the DPH will be 0000H.

**software:** Programs that contain machine-readable instructions, as opposed to hardware, which is the actual physical components of a computer.

**source file:** ASCII text file usually created with an editor that is an input file to a system program, such as a language translator or text formatter.

**SP:** Stack pointer. See **stack**.

**spooling:** Process of accumulating printer output in a file while the printer is busy. The file is printed when the printer becomes free; a program does not have to wait for the slow printing process.

**SPT:** See **sectors per track**.

**stack:** Reserved area of memory where the processor saves the return address when a call instruction is received. When a return instruction is encountered, the processor restores the current address on the stack to the program counter. Data such as the contents of the registers can also be saved on the stack. The push instruction places data on the stack and the pop instruction removes it. An item is pushed onto the stack by decrementing the stack pointer (SP) by 2 and writing the item at the SP address. In other words, the stack grows downward in memory.

**syntax:** Format for entering a given command.

**SYS:** See **system attribute**.

**SYSGEN image:** Memory image of the CP/M system created by SYSGEN when a destination drive is not specified. This is the same as the MOVCPM image that can be read by SYSGEN if a source drive is not specified. See **MOVCPM image**.

**system attribute (SYS):** File attribute. You can give a file the system attribute by using the SYS option in the STAT command or by using the set file attributes function, BDOS Function 12. A file with the SYS attribute is not displayed in response to a DIR command. If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. Use this feature to make your commonly used programs available under any user number.

**system prompt:** Symbol displayed by the operating system indicating that the system is ready to receive input. See **prompt** and **CP/M prompt**.

**system tracks:** Tracks reserved on the disk for the CP/M system. The number of system tracks is specified by the parameter OFF in the disk parameter block (DPB). The system tracks for a drive always precede its data tracks. The command SYSGEN copies the CP/M system from the system tracks to memory, and vice versa. The standard SYSGEN utility copies 26 sectors from track 0 and 26 sectors from track 1. When the system tracks contain additional sectors or tracks to be copied, a customized SYSGEN must be used.

**terminal:** See console.

**TPA:** Transient Program Area. Area in memory where user programs run and store data. This area is a region of memory beginning at 0100H and extending to the base of the CP/M system in high memory. The first module of the CP/M system is the CCP, which can be overwritten by a user program. If so, the TPA is extended to the base of the CP/M BDOS module. If the CCP is overwritten, the user program must terminate with either a system reset (Function 0) call or a jump to location zero in page zero. The address of the base of the CP/M BDOS is stored in location 0006H in page zero least significant byte first.

**track:** Data on the disk media is accessed by combination of track and sector numbers. Tracks form concentric rings on the disk; the standard IBM single-density disks have 77 tracks. Each track consists of a fixed number of numbered sectors. Tracks are numbered from zero to one less than the number of tracks on the disk.

**Transient Program Area:** See TPA.

**upward compatible:** Term meaning that a program created for the previously released operating system, or compiler, runs under the newly released version of the same operating system.

**USER:** Term used in CP/M and MP/M systems to distinguish distinct regions of the directory.

**user number:** Number assigned to files in the disk directory so that different users need only deal with their own files and have their own directories, even though they are all working from the same disk. In CP/M, files can be divided into 16 user groups.

**utility:** Tool. Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. The utilities are created for the convenience of programmers and users.

**vector:** Location in memory. An entry point into the operating system used for making system calls or interrupt handling.

**warm start:** Program termination by a jump to the warm start vector at location 0000H, a system reset (BDOS Function 0), or a CTRL-C typed at the keyboard. A warm start reinitializes the disk subsystem and returns control to the CP/M operating system at the CCP level. The warm start vector is simply a jump to the WBOOT entry point in the BIOS.

**WBOOT:** Entry point to a routine in the BIOS used when a warm start occurs. A warm start is performed when a user program branches to location 0000H, when the CPU is reset from the front panel, or when the user types CTRL-C. The CCP and BDOS are reloaded from the system tracks of drive A.

**wildcard characters:** Special characters that match certain specified items. In CP/M there are two wildcard characters: ? and \*. The ? can be substituted for any single character in a filename, and the \* can be substituted for the primary filename, the filetype, or both. By placing wildcard characters in filenames, the user creates an ambiguous filename and can quickly reference one or more files.

**word:** 16-bit or two-byte value, such as an address value. Although the Intel 8080 is an 8-bit CPU, addresses occupy two bytes and are called word values.

**WRITE:** Entry point to a routine in the BIOS that writes the record at the currently selected DMA address to the currently selected drive, track, and sector.

**XLT:** Logical-to-physical sector translation table located in the BIOS. SECTRAN uses XLT to perform logical-to-physical sector number translation. XLT also refers to the two-byte address in the disk parameter header at DPBASE + 0. If this parameter is zero, no sector translation takes place. Otherwise this parameter is the address of the translation table.

**ZERO PAGE:** See page zero.

*End of Appendix H*

# Appendix I

## CP/M Error Messages

Messages come from several different sources. CP/M displays error messages when there are errors in calls to the Basic Disk Operating System (BDOS). CP/M also displays messages when there are errors in command lines. Each utility supplied with CP/M has its own set of messages. The following lists CP/M messages and utility messages. One might see messages other than those listed here if one is running an application program. Check the application program's documentation for explanations of those messages.

Table I-1. CP/M Error Messages

Message	Meaning
?	DDT. This message has four possible meanings: <ul style="list-style-type: none"><li>■ DDT does not understand the assembly language instruction.</li><li>■ The file cannot be opened.</li><li>■ A checksum error occurred in a HEX file.</li><li>■ The assembler/disassembler was overlayed.</li></ul>
ABORTED	PIP. You stopped a PIP operation by pressing a key.
ASM Error Messages	
D	Data error: data statement element cannot be placed in specified data area.
E	Expression error: expression cannot be evaluated during assembly.
L	Label error: label cannot appear in this context (might be duplicate label).

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
N	Not implemented: unimplemented features, such as macros, are trapped.
O	Overflow: expression is too complex to evaluate.
P	Phase error: label value changes on two passes through assembly.
R	Register error: the value specified as a register is incompatible with the code.
S	Syntax error: improperly formed expression.
U	Undefined label: label used does not exist.
V	Value error: improperly formed operand encountered in an expression.
<b>BAD DELIMITER</b>	
STAT. Check command line for typing errors.	
<b>Bad Load</b>	
CCP error message, or SAVE error message.	
<b>Bdos Err On d:</b>	
Basic Disk Operating System error on the designated drive: CP/M replaces d: with the drive specification of the drive where the error occurred. This message is followed by one of the four phrases in the situations described below.	

Table I-1. (continued)

Message	Meaning
Bdos Err On d: Bad Sector	This message appears when CP/M finds no disk in the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and try again. This could also indicate a hardware problem or a worn or improperly formatted disk. Press ↑C to terminate the program and return to CP/M, or press RETURN to ignore the error.
Bdos Err On d: File R/O	You tried to erase, rename, or set file attributes on a Read-Only file. The file should first be set to Read-Write (R/W) with the command: STAT filespec \$R/W.
Bdos Err On d: R/O	Drive has been assigned Read-Only status with a STAT command, or the disk in the drive has been changed without being initialized with a ↑C. CP/M terminates the current program as soon as you press any key.
Bdos Err on d: Select	CP/M received a command line specifying a nonexistent drive. CP/M terminates the current program as soon as you press any key. Press RETURN or CTRL-C to recover.
Break "x" at c	ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred. <ul style="list-style-type: none"> <li>* Search failure. ED cannot find the string specified in an F, S, or N command.</li> <li>? Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.</li> </ul>

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
O	The file specified in an R command cannot be found.
>	Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.
E	Command aborted. A keystroke at the console aborted command execution.
F	Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages.
<b>CANNOT CLOSE DESTINATION FILE--{filespec}</b>	
	PIP. An output file cannot be closed. You should take appropriate action after checking to see if the correct disk is in the drive and that the disk is not write-protected.
Cannot close, R/O	
<b>CANNOT CLOSE FILES</b>	
	CP/M cannot write to the file. This usually occurs because the disk is write-protected.
	ASM. An output file cannot be closed. This is a fatal error that terminates ASM execution. Check to see that the disk is in the drive, and that the disk is not write-protected.
	DDT. The disk file written by a W command cannot be closed. This is a fatal error that terminates DDT execution. Check if the correct disk is in the drive and that the disk is not write-protected.
	SUBMIT. This error can occur during SUBMIT file processing. Check if the correct system disk is in the A drive and that the disk is not write-protected. The SUBMIT job can be restarted after rebooting CP/M.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
CANNOT READ	PIP. PIP cannot read the specified source. Reader cannot be implemented.
CANNOT WRITE	PIP. The destination specified in the PIP command is illegal. You probably specified an input device as a destination.
Checksum error	PIP. A HEX record checksum error was encountered. The HEX record that produced the error must be corrected, probably by re-creating the HEX file.
CHECKSUM ERROR LOAD ADDRESS hhhh ERROR ADDRESS hhhh BYTES READ: hhhh:	LOAD. File contains incorrect data. Regenerate HEX file from the source.
Command Buffer Overflow	SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.
Command too long	SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.
CORRECT ERROR, TYPE RETURN OR CTRL-Z	PIP. A HEX record checksum was encountered during the transfer of a HEX file. The HEX file with the checksum error should be corrected, probably by recreating the HEX file.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
<b>DESTINATION IS R/O, DELETE (Y/N)?</b>	PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y, the destination file is deleted before the file copy is done.
<b>Directory full</b>	ED. There is not enough directory space for the file being written to the destination disk. You can use the OXfilespec command to erase any unnecessary files on the disk without leaving the editor.
<b>SUBMIT</b>	SUBMIT. There is not enough directory space to write the \$\$.SUB file used for processing SUBMITs. Erase some files or select a new disk and retry.
<b>Disk full</b>	ED. There is not enough disk space for the output file. This error can occur on the W, E, H, or X commands. If it occurs with X command, you can repeat the command prefixing the filename with a different drive.
<b>DISK READ ERROR--{filespec}</b>	PIP. The input disk file specified in a PIP command cannot be read properly. This is usually the result of an unexpected end-of-file. Correct the problem in your file.

Table I-1. (continued)

Message	Meaning
DISK WRITE ERROR--{filespec}	<p>DDT. A disk write operation cannot be successfully performed during a W command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space.</p> <p>PIP. A disk write operation cannot be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space and execute PIP again.</p> <p>SUBMIT. The SUBMIT program cannot write the \$\$.SUB file to the disk. Erase some files, or select a new disk and try again.</p>
ERROR: BAD PARAMETER	PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly.
ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhh	LOAD. Displayed if LOAD cannot find the specified file or if no filename is specified.
ERROR: CANNOT CLOSE FILE, LOAD ADDRESS hhh	LOAD. Caused by an error code returned by a BDOS function call. Disk might be write-protected.
ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhh	LOAD. Cannot find source file. Check disk directory.
ERROR: DISK READ, LOAD ADDRESS hhh	LOAD. Caused by an error code returned by a BDOS function call.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
ERROR: DISK WRITE, LOAD ADDRESS hhhh	LOAD. Destination disk is full.
ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS hhhh	LOAD. The address of a record was too far from the address of the previously-processed record. This is an internal limitation of LOAD, but it can be circumvented. Use DDT to read the HEX file into memory, then use a SAVE command to store the memory image file on disk.
ERROR: NO MORE DIRECTORY SPACE, LOAD ADDRESS hhhh	LOAD. Disk directory is full.
Error on line nnn message	SUBMIT. The SUBMIT program displays its messages in the format shown above, where nnn represents the line number of the SUBMIT file. Refer to the message following the line number.
FILE ERROR	ED. Disk or directory is full, and ED cannot write anything more on the disk. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.
FILE EXISTS	You have asked CP/M to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.  REN. The new name specified is the name of a file that already exists. You cannot rename a file with the name of an existing file. If you want to replace an existing file with a newer version of the same file, either rename or erase the existing file, or use the PIP utility.

Table I-1. (continued)

Message	Meaning
File exists, erase it	ED. The destination filename already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file.
** FILE IS READ/ONLY **	ED. The file specified in the command to invoke ED has the Read-Only attribute. Ed can read the file so that the user can examine it, but ED cannot change a Read-Only file.
File Not Found	CP/M cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.  ED. ED cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.  STAT. STAT cannot find the specified file. The message might appear if you omit the drive specification. Check to see if the correct disk is in the drive.
FILE NOT FOUND--{filespec}	PIP. An input file that you have specified does not exist.
Filename required	ED. You typed the ED command without a filename. Reenter the ED command followed by the name of the file you want to edit or create.
hhhh??=dd	DDT. The ?? indicates DDT does not know how to represent the hexadecimal value dd encountered at address hhhh in 8080 assembly language. dd is not an 8080 machine instruction opcode.

Table I-1. (continued)

Message	Meaning
Insufficient memory	DDT. There is not enough memory to load the file specified in an R or E command.
Invalid Assignment	STAT. You specified an invalid drive or file assignment, or misspelled a device name. This error message might be followed by a list of the valid file assignments that can follow a filename. If an invalid drive assignment was attempted the message Use: d:=RO is displayed, showing the proper syntax for drive assignments.
Invalid control character	SUBMIT. The only valid control characters in the SUBMIT files of the type SUB are ^ A through ^ Z. Note that in a SUBMIT file the control character is represented by typing the circumflex, ^, not by pressing the control key.
INVALID DIGIT--{filespec}	PIP. An invalid HEX digit has been encountered while reading a HEX file. The HEX file with the invalid HEX digit should be corrected, probably by recreating the HEX file.
Invalid Disk Assignment	STAT. Might appear if you follow the drive specification with anything except =R/O.
INVALID DISK SELECT	CP/M received a command line specifying a nonexistent drive, or the disk in the drive is improperly formatted. CP/M terminates the current program as soon as you press any key.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
INVALID DRIVE NAME (Use A, B, C, or D)	SYSGEN. SYSGEN recognizes only drives A, B, C, and D as valid destinations for system generation.
Invalid File Indicator	STAT. Appears if you do not specify RO, RW, DIR, or SYS.
INVALID FORMAT	PIP. The format of your PIP command is illegal. See the description of the PIP command.
INVALID HEX DIGIT LOAD ADDRESS hhhh ERROR ADDRESS hhhh BYTES READ: hhhh	LOAD. File contains incorrect HEX digit.
INVALID MEMORY SIZE	MOVCPM. Specify a value less than 64K or your computer's actual memory size.
INVALID SEPARATOR	PIP. You have placed an invalid character for a separator between two input filenames.
INVALID USER NUMBER	PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

Table I-1. (continued)

Message	Meaning
n?	USER. You specified a number greater than fifteen for a user area number. For example, if you type USER 18<cr>, the screen displays 18?.
NO DIRECTORY SPACE	ASM. The disk directory is full. Erase some files to make room for PRN and HEX files. The directory can usually hold only 64 filenames.
NO DIRECTORY SPACE--{filespec}	PIP. There is not enough directory space for the output file. You should either erase some unnecessary files or get another disk with more directory space and execute PIP again.
NO FILE--{filespec}	DIR, ERA, REN, PIP. CP/M cannot find the specified file, or no files exist.  ASM. The indicated source or include file cannot be found on the indicated drive.  DDT. The file specified in an R or E command cannot be found on the disk.
NO INPUT FILE PRESENT ON DISK	DUMP. The file you requested does not exist.
No memory	There is not enough (buffer?) memory available for loading the program specified.

Table I-1. (continued)

Message	Meaning
NO SOURCE FILE ON DISK	SYSGEN. SYSGEN cannot find CP/M either in CP/Mxx.COM form or on the system tracks of the source disk.
NO SOURCE FILE PRESENT	ASM. The assembler cannot find the file you specified. Either you mistyped the file specification in your command line, or the filetype is not ASM.
NO SPACE	SAVE. Too many files are already on the disk, or no room is left on the disk to save the information.
No SUB file present	SUBMIT. For SUBMIT to operate properly, you must create a file with filetype of SUB. The SUB file contains usual CP/M commands. Use one command per line.
NOT A CHARACTER SOURCE	PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.
** NOT DELETED **	PIP. PIP did not delete the file, which might have had the R/O attribute.
NOT FOUND	PIP. PIP cannot find the specified file.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
OUTPUT FILE WRITE ERROR	ASM. You specified a write-protected disk as the destination for the PRN and HEX files, or the disk has no space left. Correct the problem before assembling your program.
Parameter error	SUBMIT. Within the SUBMIT file of type sub, valid parameters are \$0 through \$9.
PARAMETER ERROR, TYPE RETURN TO IGNORE	SYSGEN. If you press RETURN, SYSGEN proceeds without processing the invalid parameter.
QUIT NOT FOUND	PIP. The string argument to a Q parameter was not found in your input file.
Read error	TYPE. An error occurred when reading the file specified in the type command. Check the disk and try again. The STAT filespec command can diagnose trouble.
READER STOPPING	PIP. Reader operation interrupted.
Record Too Long	PIP. PIP cannot process a record longer than 128 bytes.
Requires CP/M 2.0 or later	XSUB. XSUB requires the facilities of CP/M 2.0 or newer version.

Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
Requires CP/M 2.0 or new for operation	PIP. This version of PIP requires the facilities of CP/M 2.0 or newer version.
START NOT FOUND	PIP. The string argument to an S parameter cannot be found in the source file.
SOURCE FILE INCOMPLETE	SYSGEN. SYSGEN cannot use your CP/M source file.
SOURCE FILE NAME ERROR	ASM. When you assemble a file, you cannot use the wildcard characters * and ? in the filename. Only one file can be assembled at a time.
SOURCE FILE READ ERROR	ASM. The assembler cannot understand the information in the file containing the assembly-language program. Portions of another file might have been written over your assembly-language file, or information was not properly saved on the disk. Use the TYPE command to locate the error. Assembly-language files contain the letters, symbols, and numbers that appear on your keyboard. If your screen displays unrecognizable output or behaves strangely, you have found where computer instructions have crept into your file.
SYNCHRONIZATION ERROR	MOVCPM. The MOVCPM utility is being used with the wrong CP/M system.
"SYSTEM" FILE NOT ACCESSIBLE	You tried to access a file set to SYS with the STAT command.

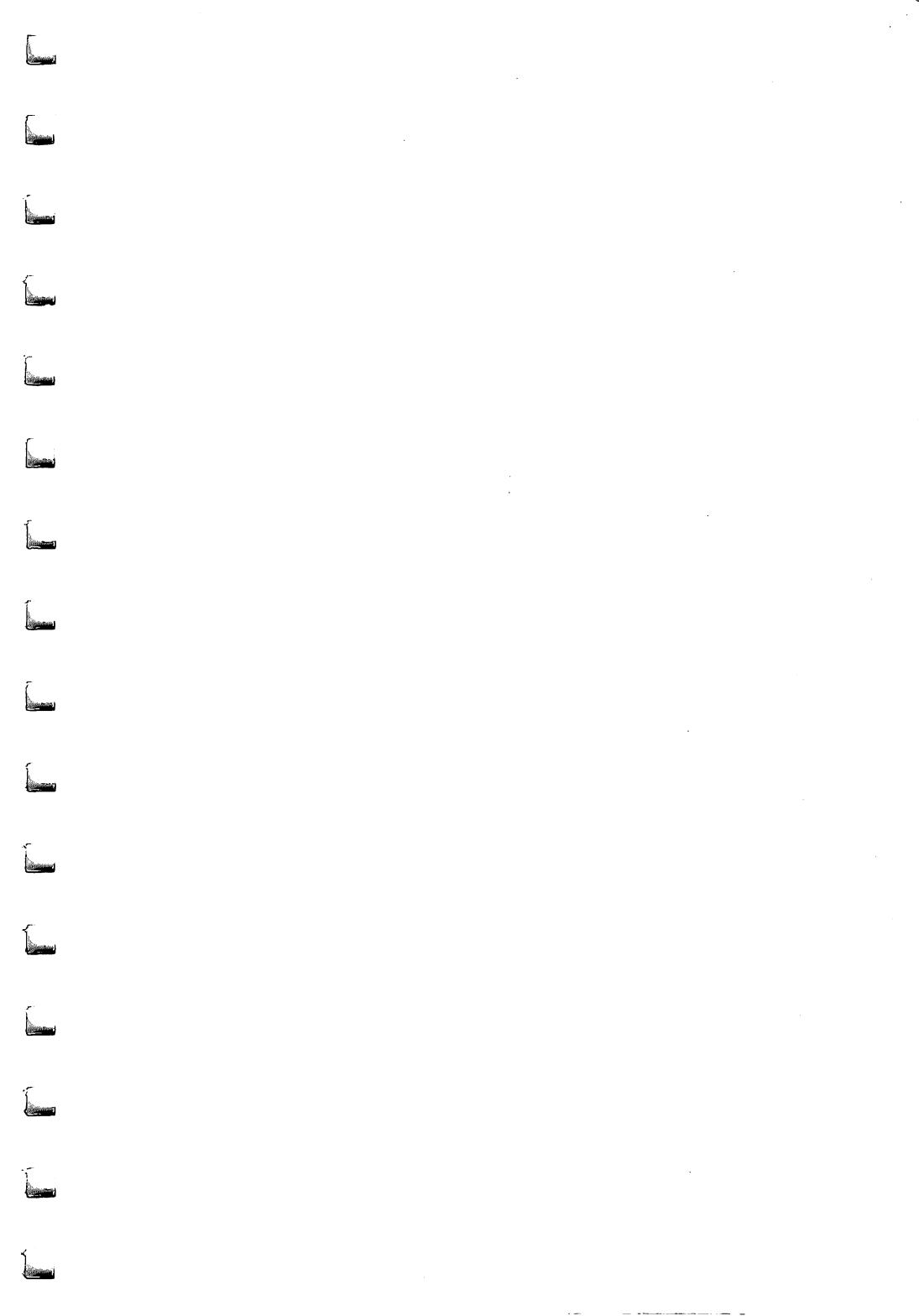
Table I-1. (continued)

<i>Message</i>	<i>Meaning</i>
<b>** TOO MANY FILES **</b>	STAT. There is not enough memory for STAT to sort the files specified, or more than 512 files were specified.
<b>UNEXPECTED END OF HEX FILE--{filespec}</b>	PIP. An end-of-file was encountered prior to a termination HEX record. The HEX file without a termination record should be corrected, probably by recreating the HEX file.
<b>Unrecognized Destination</b>	PIP. Check command line for valid destination.
<b>Use: STAT d:=RO</b>	STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:=RO.
<b>VERIFY ERROR:--{filespec}</b>	PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination disk or drive.
<b>WRONG CP/M VERSION (REQUIRES 2.0)</b>	
<b>XSUB ACTIVE</b>	SUBMIT. XSUB has been invoked.
<b>XSUB ALREADY PRESENT</b>	SUBMIT. XSUB is already active in memory.

Table I-1. (continued)

Message	Meaning
Your input?	If CP/M cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command line correctly, or that the command you requested exists as a .COM file on the default or specified disk.

*End of Appendix I*



**Appendix J  
CP/M on the BBC Microcomputer  
Technical Notes**

### **Screen Control**

There are three techniques that a CP/M application program can use to control the BBC Microcomputer's screen:

BBC Microcomputer Control Codes  
Terminal Emulator Control Codes  
GSX Functions

#### **BBC Microcomputer Control Codes**

All of the functions of the BBC Microcomputer normally accessed via the VDU command can be accessed through CP/M by sending an appropriate control code. These are explained in chapter 34 of the BBC Microcomputer User Guide and are summarised on page 378. For example, sending the sequence (as hexadecimal bytes) 1F 10 04 would position the cursor to cell x=16, y=4.

#### **Terminal Emulator Control Codes**

To allow existing CP/M applications to use basic terminal functions in a simple way, a terminal interface has been defined. This is by default disabled, but can be enabled, disabled or tested by assembler programs as follows:

To enable terminal mode

```
LD    A,1  
CALL  OFFC8H
```

To disable terminal mode

```
LD    A,0  
CALL  OFFC8H
```

To test terminal mode

```
LD    A,0FFH  
CALL  FFC8H
```

In all cases, the state of the terminal mode before the call is returned in A:

A = 0 terminal mode was disabled  
A = 1 terminal mode was enabled

An extra program is provided on the utilities disc: TERM.COM. This turns terminal mode on or off from the CCP.

TERM ON to enable terminal mode  
TERM OFF to disable terminal mode

When the terminal mode is enabled, the following control codes and escape sequences can be used to control the screen. All numbers are hexadecimal.

07	Bleep
08	Move left one character
09	Move right one character
0A	Move down one line
0B	Move up one line
0C	Clear screen
0D	Carriage return
1B 3D YY XX	Position cursor to (XX-20, YY-20)
1B 3E ?...? 00	Send sequence of bytes X...X to the screen, where each byte X sent = ?-20
1B 3F	Clear to end of screen
1B 40	Clear to end of line
1E	Home cursor

#### Notes:

To send escape (1B) to the screen with the terminal emulator enabled, the sequence 1B 3E 3B 00 should be sent.

The clear to end of line and clear to end of screen functions are intended for 80 column screens of full dimensions, and will work in screen modes 0 and 3 only. They will reset the text window to the full screen.

#### GSX Functions

These are described in the GSX Addendum.

### Operating system calls

The operating system calls of the BBC Microcomputer can be accessed directly from the Z80 in a similar manner to the BBC Microcomputer itself. Operating system calls can be made via a jump table starting at address FFCEH. The entry point for each routine corresponds with the equivalent address on the 6502, eg the WRCH routine is entered at FFEEH. All operating system calls (apart from OSARGS - see below) take parameters in Z80 registers A, H and L corresponding to A, Y and X on the 6502. For all calls that use the carry flag on the 6502 this still applies on the Z80.

For example:

```
LD A,41H      ;Character to be written in A
CALL 0FFEEH    ;Call OSWRCH to write character
```

and

```
LD A,5        ;*FX 5,2 => A set to 5
LD L,2        ;L set to 2
CALL 0FFF4H    ;Call OSBYTE routine equivalent to *FX 5,2
```

Interception of any operating system call can be achieved by simply changing the address field of the relevant jump to point to the required user routine.

The new memory map is shown below

Address (Hex)	Purpose
FFFE	INT vector - reserved for the Z80 operating system
FFFC	Event vector
FFFA	BRK vector
FFF7	OSCLI - H,L point to command line
FFF4	OSBYTE - A = OSBYTE number H,L are parameters
FFF1	OSWORD - A = OSWORD number H,L point to control block
FFEE	OSWRCH - A = character
FFE7	OSNEWL - Write linefeed, carriage return to screen
FFB3	OSASCII - Write character in A to screen plus linefeed if A = 0DH
FFE0	OSRDCH - A = character

FFDD	OSFILE - A = Operation type H,L point to control block
FFDA	OSARGS - A = Operation type , E = Handle H,L point to control block
FFD7	OSBGET - A = Byte H = File handle
FFD4	OSBPUT - A = Byte H = File handle
FFD1	OSBGBP - A = Operation type HL point to control block
FFCE	OSFIND - A = Operation type H,L point to filename (A≠0) H = file handle (A=0)
FFC8	TERM - A = 0 Switch off terminal mode (default) A = 1 Switch on terminal mode A = FF Test terminal mode
FF82	Fault pointer
FF80	Escape flag - top bit set if escape condition exists

## Faults and events

### 6502 Faults

When a fault is generated by the 6502 host processor the Z80 is interrupted and the fault number and string are passed across the Tube and placed in an fault buffer. The pointer at FF82H is then set to point to the fault number and the Z80 operating system indirects through the BRK vector at FFFAH.

### Z80 Faults

Faults can also be generated on the Z80 using the RST 38H instruction. All Z80 generated faults should adhere to the following convention:

RST 38H	Value FFH
Fault number	
Fault string	
Terminator	Value 00H

### Events

When an event is detected by the 6502 operating system the event parameters A , Y and X are passed across the Tube to Z80 registers A , H and L respectively. The Z80 operating system then indirects through the event vector at FFFCH which is initialised to point to a Z80 RET instruction.

### Escape processing

When the escape code is detected from the keyboard the top bit of the escape flag at FF80H is set. An escape condition should be detected by testing this bit and acknowledged by OSBYTE call 7EH. The escape flag should be reset or set using OSBYTE calls 7CH or 7DH.

### Interrupt handling

#### NMI Non-maskable interrupt

This interrupt is reserved for use by the Z80 operating system and cannot be intercepted by the user.

#### INT Interrupt request

When an INT is detected the Z80 operating system indirects through location FFFEH. All unrecognised interrupts are passed to a user INT routine at FFB0H in the jump table. The address field at FFB1H should be changed to point to the required user INT routine. This routine must preserve all registers and return using instructions:

```
EI ; enable interrupts  
RETI ; return from maskable interrupt routine
```

**Character I/O under CP/M****Device assignments**

The object of this implementation is to allow the user to redirect I/O either with the IOBYTE as on a normal CP/M system, or with OSBYTE calls as on a normal BBC machine.

The following logical devices are present on a CP/M system:

CON: the user console  
LST: the printer  
RDR: the paper tape reader  
PUN: the paper tape punch

These logical devices can be assigned to the following physical devices:

UC1: the user defined console device  
CRT: the screen and keyboard  
TTY: the RS423 serial lines  
LPT: the printer  
BAT: batch mode (input from RDR; and output to LST:  
PTR: paper tape reader - reassigned as the keyboard  
PTP: paper tape punch - reassigned as the screen

CP/M also has the following physical devices which are all defined as null devices in this implementation:

UR1:  
UR2:  
UP1:  
UP2:  
ULL:

Null devices throw away any output and return end of file (1AH) on input.

The default setting of IOBYTE has the following assignments:

CON: is UC1:  
LST: is LPT:  
RDR: is TTY:  
PUN: is TTY:

**Device characteristics**

The physical devices have the following characteristics:

UC1: the user defined console device.  
This is the default console device. It is also the fastest of the console devices since it uses the standard BBC input and output streams. These streams can be altered by using OSBYTE calls in the

normal BBC manner, so if you don't want to use the IOBYTE facility you can treat the machine's I/O as that of a normal BBC machine, using the STAR command.

CRT: the screen and keyboard.

Here input is taken from the keyboard and output is sent to the screen. When using this device it's characteristics cannot be changed with OSBYTE calls.

TTY: the RS423 serial lines.

The RS423 serial lines can be accessed using this device. The default baud rate setting for the TTY: is 9600 baud (this can be changed by the appropriate OSBYTE call). No events are generated by ESCAPE characters, and non-ASCII codes cannot be programmed on the function keys. The normal BBC handshaking using CTS RTS is implemented. You should not enable the cassette drivers or disable the RS 423 input while using this device. Although a serial printer can be driven by this device you should be aware that this bypasses the normal printer functions, such as setting an ignore character or being informed that the printer has stopped. It is therefore better to use the LPT: device and set it to a serial printer with the appropriate OSBYTE call.

LPT: the printer.

This is the standard BBC printer driver. It can be changed to suit the particular printer in use. The default setting is the parallel printer, ignoring linefeeds. It is recommended that when connecting a serial printer the IOBYTE be left unchanged and the LPT: device altered with the appropriate OSBYTE call. The alternative is to use the IOBYTE to select the TTY: driver but this does not carry out any of the standard printer functions.

BAT: batch mode (input from RDR: and output to LST:).

This device takes its input from the logical device RDR:. This can in turn be assigned to any of the relevant physical devices. The output goes to the logical device LST:, which can in turn be reassigned to the relevant physical device.

PTR: paper tape reader - reassigned as the keyboard.

In the absence of a paper tape reader this device is the keyboard.

PTP: paper tape punch - reassigned as the screen.

In the absence of a paper tape punch this device is the screen.

**Boot options on the BBC CP/M system**

The BBC CP/M system is provided with an option to allow automatic execution of programs on initial startup. The system checks disc A, user area 0, for the presence of certain files and will take appropriate action if these are found. On subsequent warm starts no action is taken.

There are two possible types of BOOT file; BOOT.COM and BOOT.SUB. BOOT.COM takes precedence. If it is present then it is run as a normal .COM program. If there is no BOOT.COM but there is a BOOT.SUB then this will be submitted. If both BOOT.COM and BOOT.SUB are present then the BOOT.COM is run and the BOOT.SUB is ignored.

BOOT.COM is a normal CP/M COM file, ie. a machine code program that is loaded and run directly. If this is present then it is loaded and run before the initial prompt. An example of this would be to rename STAT.COM to BOOT.COM and so be informed of the free space on the disc when starting up. Another use would be to start running an applications program immediately so that the users never need deal directly with the CP/M operating system.

BOOT.SUB is a normal CP/M SUBMIT file, ie. it is a text file containing a list of CP/M commands to be executed. If BOOT.COM isn't present then CP/M looks for BOOT.SUB. If it is found then CP/M attempts to submit it. To submit it the file SUBMIT.COM must be present on disc A, user 0. If it is missing then a message to that effect will be given and the BOOT option ignored.

# Index

## A

Absolute line number, 2-5  
Access mode, 1-19  
afn (ambiguous file reference), 1-4, 1-7  
Allocation vector, 5-27  
Ambiguous file reference (afn), 1-4, 1-7  
ASM, 1-22, 3-1  
Assembler, 1-22, 3-1  
Assembler/disassembler module (DDT), 4-11  
Assembler errors, 3-24  
Assembly language mnemonics in DDT, 4-4, 4-7  
Assembly language program, 3-3  
Assembly language statement, 3-3  
Automatic command processing, 1-39

## B

Base, 3-5  
Basic Disk Operating System (BDOS), 1-2, 5-1, 6-1  
Basic I/O System (BIOS), 1-2, 5-1, 6-1  
BDOS (Basic Disk Operating System), 1-2, 5-1, 6-1  
Binary constants, 3-5  
BIOS (Basic I/O System), 1-2, 5-1, 6-1  
BIOS disk definition, 6-34  
BIOS subroutines, 6-15  
Block move command, 4-8  
bls parameter, 6-35  
BOOT, 5-2, 6-13, 6-20  
BOOT entry point, 6-20  
Break point, 4-4, 4-6  
Built-in commands, 1-3

## C

Case translation, 1-6, 1-7, 1-31, 1-32  
1-33, 2-7, 2-10, 2-20, 2-21, 2-22,  
3-7, 5-10, 5-11  
CCP (Console Command Processor),  
1-2, 4-1, 5-1, 6-1  
CCP Stack, 5-6  
Character pointer, 2-4  
cks parameter, 6-35  
Close File function, 5-20  
Code and data areas, 6-26  
Cold start loader, 6-13, 20, 25  
Command, 1-3  
Command line, 5-3  
Comment field, 3-4  
Compute File Size function, 5-33  
Condition flags, 3-17, 4-11  
Conditional assembly, 3-14  
CONIN, 6-21  
CONOUT, 6-21  
CONSOLE, 6-18  
Console Command Processor (CCP),  
1-2, 4-1, 5-1, 6-1  
Console Input function, 5-12  
Console Output function, 5-12  
CONST, 6-21  
Constant, 3-5  
Control characters, 2-19  
Control functions, 1-13  
CTRL-Z character, 5-7  
Copy files, 1-25  
CPU state, 4-3, 4-4  
cr (carriage return), 2-10  
Create files, 1-35  
Create system disk, 1-37  
Creating COM files, 1-24  
Currently logged disk, 1-3, 1-7, 1-15, 1-36

## D

Data allocation size, 6-31  
Data block number, 6-32  
DB statement, 3-15  
DDT commands, 4-4, 6-9  
DDT nucleus, 4-11  
DDT prompt, 4-2  
DDT sign-on message, 4-1  
Decimal constant, 3-5  
Default FCB, 4-7  
Delete File function, 5-22  
DESPOLL, 6-17  
Device assignment, 1-16  
DIR, 1-9  
DIR attribute, 1-20  
dir parameter, 6-35  
Direct console I/O function, 5-14  
Direct Memory Address, 5-27  
Directory, 1-9  
Directory code, 5-19, 5-20, 5-21, 5-22, 5-23, 5-24, 5-25  
Disassembler, 4-4, 11  
Disk attributes, 1-15  
Disk drive name, 1-6, 7  
Disk I/O functions, 5-17—5-35  
Disk parameter block, 6-30  
Disk parameter header, 6-28  
Disk parameter table, 6-28  
Disk statistics, 1-15  
Disk-to-disk copy, 1-27  
DISKDEF macro, 6-34  
Diskette format, 1-47  
DISKS macro, 6-34  
Display file contents, 1-11  
dks parameter, 6-35  
DMA, 5-27  
DMA address, 5-8  
dn parameter, 6-35  
DPBASE, 6-29

Drive characteristics, 1-21  
Drive select code, 5-9  
Drive specification, 1-7  
DS statement, 3-16  
DUMP, 1-41 5-40  
DW statement, 3-15

## E

ED, 1-35, 2-1—2-22, 6-6  
ED commands, 2-8, 19  
ED errors, 2-18  
Edit command line, 1-12  
8080 CPU registers, 4-10  
8080 registers, 3-6  
end-of-file, 1-28, 5-7  
END statement, 3-4, 3-11  
EMDEF macro, 6-35  
ENDIF statement, 3-13  
EQU statement, 3-12  
ERA, 1-8  
Erase files, 1-8  
Error messages, 1-44, 2-18, 3-24  
Expression, 3-4  
Extents, 1-19

## F

FBASE, 5-2  
FCB, 5-8, 5-9  
FCB format, 5-8, 5-9  
FDOS (operations), 5-1, 5-4  
File attributes, 1-20  
File compatibility, 1-35  
File control block (FCB), 5-8, 5-9  
File expansion, 6-2  
File extent, 5-8  
File indicators, 1-20  
File names, 1-4

File reference, 1-4  
File statistics, 1-15, 1-19  
Filetype, 5-6  
Find command, 2-11  
fsc parameter, 6-35

## G

Get ADDR (Alloc) function, 5-27  
Get ADDR (DiskParms) function, 5-29  
Get Console Status, 5-17  
Get I/O Byte function, 5-15  
Get Read/Only Vector function, 5-28  
GETSYS, 6-3, 6-11

## H

Hexadecimal constant, 3-5  
HOME subroutine, 6-20, 22

## I

Identifier, 3-3, 3-5  
IF statement, 3-13  
Initialized storage areas, 3-15  
In-line assembly language, 4-4  
Insert mode, 2-7  
Insert String, 2-12  
IOBYTE function, 6-17—6-19

## J

Jump vector, 6-15  
Juxtaposition command, 2-15

## K

Key fields, 5-34

## L

Label field, 3-3  
Labels, 3-3, 3-4, 3-16  
Library read command, 2-16  
Line-editing control characters, 2-9, 4-2, 5-16  
Line-editing functions, 1-12  
Line numbers, 2-5  
LIST, 6-17, 6-21  
List Output function, 5-14  
LISTST, 6-24  
LOAD, 1-24  
Logged in, 1-3  
Logical devices, 1-16, 1-28, 6-17  
Logical extents, 5-8  
Logical-physical assignments, 1-18, 6-19  
Logical to physical device mapping, 6-18  
Logical to physical sector translation, 6-24, 6-35  
Isc parameter, 6-35

## M

Macro command, 2-17  
Make File function, 5-25  
Memory buffer, 2-1—2-7  
Memory image, 4-3, 6-6, 6-7  
Memory size, 1-42, 6-3, 6-8  
MOVCPM, 1-42, 6-7

## N

Negative bias, 6-7

## O

[o] parameter, 6-35

Octal constant, 3-5

ofs parameter, 6-35

On-line status, 5-19

Open File function, 5-19

Operand field, 3-4, 3-6

Operation field, 3-4, 3-16

Operators, 3-9, 3-16

ORG directive, 3-11

## P

Page zero, 6-26

Patching the CP/M system, 6-3

Peripheral devices, 6-17

Physical devices, 1-17, 6-17

Physical file size, 5-33

Physical to logical device assignment, 1-18, 6-19

PIP devices, 1-28

PIP parameters, 1-31

Print String function, 5-15

PRN file, 3-1

Program counter, 4-4, 4-6, 4-7, 4-11

Program tracing, 4-9

Prompt, 1-3

Pseudo-operation, 3-10

PUNCH, 6-17, 6-21

Punch Output function, 5-13

PUTSYS, 6-4, 6-11

## R

Radix indicators, 3-5

Random access, 5-31, 5-32, 5-46

Random record number, 5-32

READ, 6-23

Read Console Buffer function, 5-16

Read only, 1-20

Read/only status, 1-20

Read random error codes, 5-31

Read Random function, 5-30

READ routine, 6-20

Read Sequential function, 5-23

Read/write, 1-20

READER, 6-18, 21

Reader Input function, 5-13

REN, 1-10

Rename file function, 5-25

Reset Disk function, 5-18

Reset Drive function, 5-35

Reset state, 5-18

Return Current Disk function, 5-26

Return Log-in Vector function, 5-26

Return Version Number function, 5-18

R/O, 1-20

R/O attribute, 5-29

R/O bit, 5-28

R/W, 1-20

## S

SAVE, 1-11

SAVE command, 4-3

Search for First function, 5-21

Search for Next function, 5-22

Search strings, 2-11

Sector allocation, 6-13

SECTRAN, 6-24  
SELDSK, 6-19, 6-22, 6-30  
Select Disk function, 5-19  
Sequential access, 5-8  
Set DMA address function, 5-27  
Set File Attributes function, 5-29  
Set/Get User Code function, 5-30  
Set I/O Byte function, 5-15  
Set Random Record function, 5-34  
SET statement, 3-13  
SETDMA, 6-23  
SETSEC, 6-23  
SETTRK, 6-22  
Simple character I/O, 6-17  
Size in records, 1-19  
skf parameter, 6-35, 6-37  
Source files, 5-7  
Stack pointer, 5-6  
STAT, 1-15, 6-17, 6-38  
Stop console output, 1-13  
String substitutions, 2-14  
SUBMIT, 1-39  
SYS attribute, 1-20  
SYSGEN, 1-37, 6-10  
System attribute, 2-19, 5-29  
System parameters, 6-20  
System (re)initialization, 6-16  
System Reset function, 5-11

## T

Testing and debugging of programs, 4-1  
Text transfer commands, 2-3  
TPA (Transient Program Area), 1-2, 5-1  
Trace mode, 4-10  
Transient commands, 1-3, 1-14  
Transient Program Area (TPA), 1-2, 5-1

Translate table, 6-37  
Translation vectors, 6-30  
TYPE, 1-11

## U

ufn, 1-4, 1-7  
Unambiguous file reference, 1-4, 1-7  
Uninitialized memory, 3-16  
Untrace mode, 4-10  
USER, 1-12  
USER numbers, 1-12, 1-22, 5-30

## V

Verify line numbers command, 2-6, 21  
Version independent programming, 5-18  
Virtual file size, 5-33

## W

Warm start, 5-2, 6-20  
WBOOT entry point, 6-20  
WRITE, 6-24  
Write Protect Disk function, 5-28  
Write random error codes, 5-32  
Write Random function, 5-32  
Write Random with Zero Fill function, 5-35  
Write routine, 6-24  
Write Sequential function, 5-24

## X

XSUB, 1-41

## **NOTES**

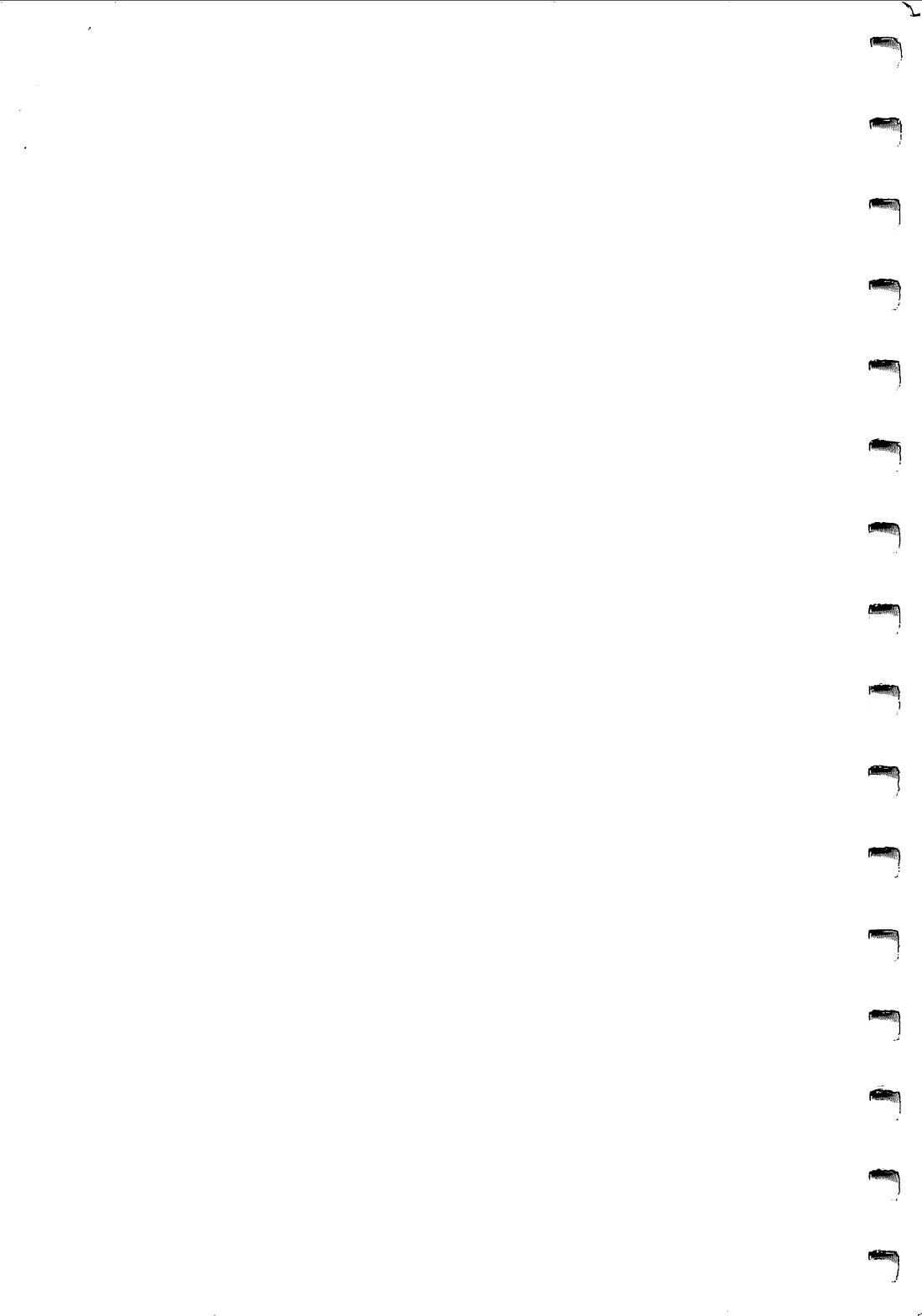
**ADDENDUM**  
**GSX GRAPHICS PACKAGE**  
**USER GUIDE**

Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.



## **CONTENTS**

	<b>Introduction</b>	1
1	<b>What is GSX?</b>	3
	The user's view	3
	The programmer's view	3
2	<b>How to use GSX</b>	5
	Setting up GSX	5
	Invoking GSX	6
3	<b>Writing application programs to use GSX</b>	7
	The GDOS	7
	GDOS calls	8
	Input	8
	Output	8
	GDOS fuctions	9
4	<b>GSX drivers provided</b>	11



## **Introduction**

GSX-80 adds graphics capability to the CP/M system running on your BBC Microcomputer. GSX stands for Graphics System Extension, and is just that. It works in the same way as the CP/M operating system and allows graphics programs to be transferred unchanged from machine to machine.

This guide is designed to provide an overview of the features of GSX-80 on the BBC Microcomputer, and how these features might be used. GSX has been provided to allow access to commercially written graphics applications. You should use this guide in conjunction with support documentation for the applications you wish to use.

Various programming languages available to run under CP/M have a built-in interface to GSX-80. Programs written in these languages will transfer directly to the BBC Microcomputer with Z80 second processor. Further details will be found in the appropriate language reference manual. There are chapters later in this guide on using GSX from other languages. Technical details of GSX-80 will be found in the Digital Research Manual, GSX-80 Programmers Guide.

2

1

1

三

1

1

1

—

1

1

1

3

1

1

f  
-

## **1 What is GSX?**

### **The user's view**

Not all the graphics input and output devices which can be used with the BBC Microcomputer have the same facilities or capabilities. A simple task to the BBC Microcomputer such as drawing a solid red triangle on a blue background is difficult to accomplish on a colour plotter and impossible on a 'standard' matrix printer! If GSX is used by your application programs, then it will handle the differences and simulate (within reason) the features the device cannot provide. Even with GSX there are limits to this simulation capability, so you should choose the features required with the devices in mind.

GSX consists of a set of programs, one per device, to translate your device-independent operators into device-specific commands. These are called 'device drivers'. For example, although every device has a co-ordinates system of some kind, direct access to co-ordinates in the range X=0 1279 and Y=0 to 1023 on the BBC Microcomputer and vertical and horizontal increments on a printer, there is very little standardisation. The device drivers ensure that your problem oriented co-ordinates system is translated into the appropriate co-ordinates for the device being used, so ensuring correctly positioned text of the right size and correct orientation.

In this way, you leave it up to GSX to fetch the correct device driver from disc and load it into the memory of the Z80, ready to obey the application program's graphics commands. In return, however, you have to add a few special instructions to your programs or CP/M command sequences to inform GSX which device you are actually using. GSX does not assume that a graphics device is simply an output device (such as a plotter), but if required any device can also be used for input. GSX categorises input devices as follows:

Locator	- 2-D position eg Mouse, Light Pen, Joystick
Valuator	- Analog 1-D value (eg A to D converter)
Choice	- Selection from a set of values (eg Cursor keys)
Input String	- Keyboard input

### **The programmer's view**

GSX consists of the device-independent Graphics Device Operating System (GDOS) and the device-dependent Graphics Input/Output System (GIOS). The graphics application makes calls on the GDOS. The GDOS both loads the appropriate device driver into the GIOS area and makes low level function calls on the GIOS.

A graphics application program runs in the Transient Program Area (TPA), just like any other user program, but first reserves space for the GIOS and loads the GDOS at the top of the TPA. Rather than include the GIOS device drivers and the GDOS in each application program, these are loaded from disc as needed. The GENGRAF utility is used to incorporate the GSX loader in the application program. The loader runs at the start of the application and loads in the GDOS from the file GSX.SYS and the list of device drivers from ASSIGN.SYS. The GDOS dynamically loads device drivers from disc.



## **2 How to use GSX**

### **Setting up GSX**

On the CP/M Utilities disc are some special files for GSX to use. They are:

```
ASSIGN.SYS  
GSX.SYS  
DDBBC0.PRL  
DDBBC1.PRL  
DDFXL8R  
DDFXHR8  
DDOKI84  
GENGRAF.COM
```

ASSIGN.SYS contains the Assignment Table. This 'assigns' logical device numbers to the device drivers. Application programs are written using logical devices, and so simply changing the assignment table is all that is needed to specify which real devices will be used.

GSX.SYS contains GSX itself (the GDOS).

DD---RL are the device drivers provided with the system. You should consult Acorn Computers Limited Customer Service Department if you wish to use other devices, and the Digital Research technical manuals if you wish to write your own.

GENGRAF.COM is a program which adds the GSX loader to a graphics application program. This will only be used if you are writing or modifying your own graphics applications programs.

ASSIGN.SYS contains one line per device driver assignment. Each line consists of a two-digit logical device number followed by a space and the filename of the appropriate device driver, for example

01 B:DDBBC1

assigns logical device number 01 to be the BBC Microcomputer display in Mode 1 -for which the device driver is DDBBC1.PRL from drive B.

For compatibility with other GSX systems, it is recommended that you use logical device numbers from 01 to 10 for terminals, 11-20 for plotters and 21-30 for printers. 0 is used for the default console.

It is important that the largest device driver is the first in the list in ASSIGN.SYS. Use STAT to find which this is.

To create an application disc which uses GSX:

1. Copy the required device drivers onto the application disc using the PIP utility.
2. Copy GSX.SYS
3. If necessary edit ASSIGN.SYS to match the logical numbers used in the application and the device drivers available, and copy this across too.

This disc can then be used as a GSX disc by being in one disc drive

when another disc containing GSX application programs is used; alternatively the application programs can be put onto this disc.

#### **Invoking GSX**

You should ensure that the logged in drive contains:

ASSIGN.SYS (which references the logged in drive)  
The device drivers listed in ASSIGN.SYS  
GSX.SYS

The application programs you wish to run can be on any disc, and are run simply by typing in their filenames, just like running a normal application program.

### **3 Writing application programs to use GSX**

This chapter summarises the technical details of using the device-independent graphics from an application program. It will provide an overview of how to interface to the Graphics Device Operating System (GDOS). Full details of the function calls and their parameters will be found in the Digital Research manual GSX-80 Programmers Guide.

#### **The GDOS**

The application program invokes the graphics features via calls to the GDOS. The GDOS handles the conversion of normalised co-ordinates (the device-independent co-ordinate system used by GSX) to device-dependent co-ordinates. It also loads any device drivers needed to control the actual graphics devices in use.

The application program is written just like any other CP/M application program, but contains extra calls to the BDOS to invoke the GDOS functions. The BDOS will only recognise GDOS functions if GSX has been loaded, and so an extra stage in the program development is required. After the program has been assembled or compiled and linked to produce a .COM file, the program GENGRAF is run to add the GSX loader to the application program. Then the application is ready to be run as described in section 2 of this guide.

For example: A program called DRAWIT

Write graphics application in Pascal/MT+ say	DRAWIT.SRC
Compile and Link with Runtime system giving	DRAWIT.COM
Type GENGRAF DRAWIT to add GSX loader giving new	DRAWIT.COM

The GDOS is invoked by a BDOS function call with code 115. For code 115 to be recognised, it is necessary to have started GSX from the GSX loader. The application should use Normalised Device Coordinates when using GDOS functions. These are in the range 0 to 32767 in both the X and Y axes. GDOS converts these co-ordinates to those of the device using data passed from the device driver when the graphics device was opened. The full scale co-ordinates are always mapped onto the full dimensions of the device in each axis, thus ensuring that all the graphics display is visible on the device

## **GDOS calls**

Invoked by BDOS call with function 115 (73H) and passing a parameter block.

```
LD C, 73H      ; function code
LD DE, PARAMB ; parameter block
CALL 0005H      ; BDOS call
```

The details of the call are provided in the parameter block, which contains data as follows:

Bytes	Address of:
0,1	control array (CTRL)
2,3	input parameter array (INTIN)
4,5	input point co-ordinate array (PTSIN)
6,7	output parameter array (INTOUT)
8,9	output point co-ordinate array (PTSOUT)

### **Input**

- Control array
  - CONTROL (1) opcode for driver function
  - CONTROL (2-3) number of vertices in input point array
  - CONTROL (4-5) length of input parameter array
  - CONTROL (6-...) dependent on the function
- Parameter array
  - INTIN (1-...) array of input parameters
- Co-ordinate array
  - PTSIN (1-...) array of input co-ordinates. Each point is given by an X and a Y co-ordinate (2 bytes each).  
The number of points depends on the function.

### **Output**

- Control array
  - CONTROL (3-4) number of vertices in output point array
  - CONTROL (5) length of output co-ordinate array
  - CONTROL (6-...) dependent on the function
- Parameter array
  - INTOUT (1-...) array of output parameters
- Point co-ordinate array
  - PTSOUT (1-...) array of output co-ordinates - as for input

Note any lengths and number of parameters of unused arrays must be zero.

## **GDOS functions**

- 1 Initialise a graphics device (load driver if necessary)
- 2 Stop graphics output to this workstation
- 3 Clear display device or move to top of form
- 4 Display all pending graphics on workstation
- 5 Enable device-dependent operation (eg cursor moves, clear line)
- 6 Output a polyline
- 7 Output markers
- 8 Output text, starting at specified position
- 9 Display and fill a polygon
- 10 Define a cell array
- 11 Display a special shape or character (eg bar, arc, pie slice)
- 12 Set size for text output
- 13 Set text direction
- 14 Define the colour associated with a colour index
- 15 Set polyline line type
- 16 Set polyline linewidth
- 17 Set polyline colour index
- 18 Set polymarker type
- 19 Set polymarker size
- 20 Set polymarker colour index
- 21 Set device-dependent text style
- 22 Set text colour index
- 23 Set interior style for polygon fill
- 24 Set fill style for polygons
- 25 Set colour for polygon fill
- 26 Return colour representation values of colour index
- 27 Return definition of cell array
- 28 Return value of 'Locator'
- 29 Return value of 'Valuator'
- 30 Return value of 'choice device'
- 31 Return character string input to 'Text Input Device'
- 32 Set Writing Mode
- 33 Set Input Mode

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

ר

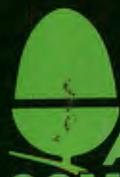
#### **4 GSX drivers provided**

Included within the BBC Microcomputer Z80 Second Processor package are device drivers for the BBC Microcomputer screen in mode 0 and 1, plus the Epson and OKidata range of printers.

### DEVICE DRIVERS

	BBC MICROCOMPUTER		PRINTERS		
DEVICE	Mode 0	Mode 1	Epson (Low Res)	Epson (High Res)	Microline
DRIVER NAME	DDBBC0	DDBBC1	DDFXLR8	DDFXHR8	DDOKI84
LOGICAL DEVICE	0	1	21	22	23
RESOLUTION	640x256	320x256	480x672	960x1368	824x672
LINESTYLES	1 Solid 2 Dashed (1.5 - 1.5) 3 Dashed (4.5 - 4.5) 4 Dashed (3 - 1.5) 5 Dashed (6 - 1.5)		1 Solid 2 Short Dash 3 Dot 4 Dash-Dot 5 Long Dash 6 Dash Dot-Dot		
NUMBER OF SIZES	1			12	
MARKER TYPES			1 . 2 + 3 * 4 0 5 X		
TEXT - SIZE - RESOLUTION	1 0		12 90 Degree Intervals		
FILL AREA			Hatch or Half-Tone 1 2 3 4 5 6		
GENERALISED DRAWING PRIMITIVES	Bar Fill Pie Slices } Solid Black/White Red/Green Blue/White Arcs } Solid Circles Black/White Red/Green Blue/White		Bar Fill Black/White		
ESCAPES	No		Inquire Addressable Character Cells		
INPUT	No		No		





**ACORN  
COMPUTER**

Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, England

Printed by Saunders & Williams (Printers) Ltd, Croydon, Surrey